HaSoTec

Frame Grabber FG-32/ FG-34 PCI
FG-31 ISA/ FG-35 Low-Profile PCI
FG-33 CardBus (32-bit-PCMCIA)
FG-30 PCMCIA

Programming examples and Information

Version 4.87

Contents

1. Programmer's guide and general remarks

This chapter describes programming on procedural level (High-Level programming) and programming with direct device driver calls (Low-Level programming).

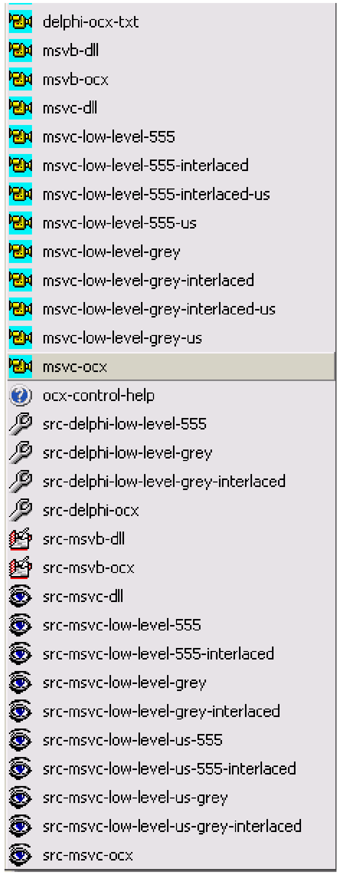
For High-Level programming with OCX Control please refer to chapter 9, where common OCX functions are described. For programming under OS/2 refer to chapter 8 for OS/2-specific functions.

In 5.1 of this chapter the API 9709 device driver calls for all operating systems will be shown. Using this interface it is possible to write program code parts that are independent from operating systems.
At the same time this API (Application Programmers Interface) is a standard interface useable for Frame Grabbers FG31... FG35. A first API 9209 for FG-30 ISA was developed in 1992. This API is supported for FG30 PCMCIA for all Windows versions up to Windows Me. API 9709, developed in 1997, contains all functions of API 9209, but it is based on 32-bit data ports instead of 16-bit. Ready-to-use applications with their libraries, DLLs and OCX Controls use the same API. Under DOS, Windows 3.0, Windows 3.1, Windows 3.11, Windows 95, Windows 98 / Me API calls are made with the help of a Software Interrupt 60H. Parameters are sent by registers ax, bx, cx and dx. Under Windows NT 3.51, Windows NT 4.0, Windows NT 5.0, Windows 2000 and Windows XP, as well as OS/2 Version 2.0 or higher and Linux, a device driver call has the same parameters, but variables are used instead of registers. For convenience, these variables have names that contain ax, bx, cx and dx. This interface is implemented in some libraries, DLLs and OCX Controls, too. FG30.OCX, for example, contains a function FG30DRV (this name is the same for all FG-3x grabbers) to make Low-Level calls. FG30.OCX detects

the operating system that is running and switches to the correct driver call.

This FG30DRV function is implemented in all OS/2 libraries.

The choice of the VGA driver used under MS-Windows 3.x-9x is important. For most applications, 32768 or 65536 (15 or 16 bit/pixel) colors make the most sense. This ensures that images can be displayed in grey scale or color at a reasonable quality without using complex palette functions. The source code examples that are supplied are complete applications which can be easily expanded. If one of the directly supported compilers is used, there are both simple and complex programs that can be adopted for different purposes. There is often no need to write your own program code to grab and display images. Frames can be captured in all video standards. Some applications have capturing with averaging and some have dialog boxes to manipulate the grabber's adjustments. All source code examples come together with their compiled *.EXE files. This makes it possible to see how the examples work even if there is no compiler installed. It makes sense to compile the chosen example and to check for differences that may appear between compiler versions. Cf. the screen shot (left): executable files have a camera symbol. Menu points with other symbols call compiler environments with their project files.



We believe that these source code examples will help you start writing your own programs under Windows quickly.
With just a few Windows API calls, such as GlobalAlloc, GlobalLock, GlobalUnlock, GlobalFree, SetDIBBitsToDevice and knowing the structure of BITMAPINFOHEADER and Device Independent Bitmaps (DIB) new users coming from different platforms will be able to solve most of their programming problems.

1.1 High-Level Programming under Windows XP/ 2000/ NT

High- Level Programming examples use OCX- Controls, DLLs, Object- or LIB- files. These files contain more complex functions than those of Low-Level examples. Typically, a single function call grabs an image or opens a complex dialog box. For some of these files it is important to install the supplied CD under the target operating system. For example, a DLL for Win98 may not be the same as the DLL for WinXP. There are internal differences even if their functionality is the same. The installation program identifies the operating system and conditionally installs the right components. Not only for High-Level programs is it important to call the correct library. It is also important for object files or DLLs to provide and install the right library for the target system.

Currently all OCX- controls for any given single Frame Grabber card can be used for all operating systems.
DLLs, libraries and object files are different for the first group of operating systems (Windows XP/ 2000/ NT) and for the second group (Windows Me/ 98/ 95). If a file from the second group is used under Windows XP/ 2000/ NT, a first I/O port access will result in an exception, which is shown as an OS Error message.

Under Windows XP/ 2000/ NT only 32-bit programs are installed

and supported. To use 16-bit source code or DOS examples, development must be carried out on a Windows Me/ 98/ 95/3.x computer.

1.2 Low-Level Programming under Windows XP/ 2000/ NT

Under Windows XP/2000/NT, Linux and OS/2 each driver call to the Frame Grabber API is made with a fixed set of variables. These variables (bx, cx, and dx) have the same parameter names Some helpful tips about using driver calls A Low-Level call to the Frame Grabber API is made by the following function:

```
IoctlResult = DeviceIoControl ( hdev,           // Handle to device
                   (ULONG)FKT020,     // IO Control code Low-Level
                   &freg,                          // Buffer to driver.
                   sizeof (FREG),,         // Length of buffer in bytes.
                   &freg,                        // Buffer from driver.
                   sizeof (FREG),          // Length of buffer in bytes.
                   &ReturnedLength,     // Bytes placed in DataBuffer
                   NULL
                     );
```

hdev is returned, when the device driver is opened:

```
hdev = CreateFile    ("\\\\.\\Fg32Dev",
                   GENERIC_READ,FILE_SHARE_READ, NULL,
                               OPEN_EXISTING, 0, NULL);
```

You should return this handle to the operating system when the program is closed using:

```
Close (hdev)
```

FREG is a data structure that contains the following components:

```
typedef struct
        {
        USHORT fnr;    //bx
        USHORT cx;
        USHORT dx;
        PUCHAR reserved;
        ULONG reserved2;
        } FREG;
typedef FREG * PFREG;
```

Section 5 describes how to use the variables fnr (bx), cx and dx. Under Windows XP, 2000 and NT there are additional constraints that have to be taken into account. Accessing I/O ports directly from the user program is not allowed. Image data is readable sequentially from I/O ports. WinMe/9x/3x and DOS Programs can directly implement functions to read image data, but WinXP/2000/NT programs need additional data transfer functions to be implemented in the Device Driver FG32DRV.SYS.
Data transfer functions read the Frame Grabber’s on-board memory and transfer the data to a pointer of a user-created data buffer or directly to VGA memory. These functions are shown below:

Pos.	FG-3x High-Leve l functions API Funktion	Grabben	Aus-lesen bits	Daten	X-Auf-löösung	Y-Auf-löösung	Norm	Kopf-stehend	Aver-aging	WinNT FNR: DevIoCtl
	Grau									
1	FG32IMG160X120X8	grbflg	8	Grey8	160	120	US	nein	nein	IMG001
2	FG32IMG192X144X8	grbflg	8	Grey8	192	144	Eu	nein	nein	IMG002
3	FG32IMG320X240X8	grbflg	8	Grey8	320	240	US	nein	nein	IMG003
4	FG32IMG384X288X8	grbflg	8	Grey8	384	288	Eu	nein	nein	IMG004
5	FG32IMG640X480X8	grbflg	8	Grey8	640	480	US	nein	nein	IMG005
6	FG32IMG768X576X8	grbflg	8	Grey8	768	576	Eu	nein	nein	IMG006
	Grau with Averaging									
7	FA32IMG160X120X8	ja	8, 16	Grey8	160	120	US	nein	ja	IMG011
8	FA32IMG192X144X8	ja	8, 16	Grey8	192	144	Eu	nein	ja	IMG012
9	FA32IMG320X240X8	ja	8, 16	Grey8	320	240	US	nein	ja	IMG013
10	FA32IMG384X288X8	ja	8, 16	Grey8	384	288	Eu	nein	ja	IMG014
11	FA32IMG640X480X8	ja	8, 16	Grey8	640	480	US	nein	ja	IMG015
12	FA32IMG768X576X8	ja	8, 16	Grey8	768	576	Eu	nein	ja	IMG016
	Grau in DIB									
13	FG32DIB160X120X8	grbflg	8 bit	Grey8	160	120	US	ja	nein	IMG021
14	FG32DIB192X144X8	grbflg	8 bit	Grey8	192	144	Eu	ja	nein	IMG022
15	FG32DIB320X240X8	grbflg	8 bit	Grey8	320	240	US	ja	nein	IMG023
16	FG32DIB384X288X8	grbflg	8 bit	Grey8	384	288	Eu	ja	nein	IMG024
17	FG32DIB640X480X8	grbflg	8 bit	Grey8	640	480	US	ja	nein	IMG025
18	FG32DIB768X576X8	grbflg	8 bit	Grey8	768	576	Eu	ja	nein	IMG026
	Color 24									
19	FG32IMG160X120X24	grbflg	24, 48	RGB	160	120	US	nein	nein	IMG031
20	FG32IMG192X144X24	grbflg	24, 48	RGB	192	144	Eu	nein	nein	IMG032
21	FG32IMG320X240X24	grbflg	24, 48	RGB	320	240	US	nein	nein	IMG033
22	FG32IMG384X288X24	grbflg	24, 48	RGB	384	288	Eu	nein	nein	IMG034
23	FA32IMG592X442X24US	grbflg	24, 48	RGB	592	442	US	nein	nein	IMG035
24	FA32IMG592X442X24	grbflg	24, 48	RGB	592	442	Eu	nein	nein	IMG036
25	FG32IMG640X480X24	grbflg	24, 48	RGB	640	480	US	nein	nein	IMG037
26	FG32IMG768X576X24	grbflg	24, 48	RGB	768	576	Eu	nein	nein	IMG038
	Color Averaging									
27	FA32IMG160X120X24	ja	24, 48	RGB	160	120	US	nein	ja	IMG041
28	FA32IMG192X144X24	ja	24, 48	RGB	192	144	Eu	nein	ja	IMG042
29	FA32IMG320X240X24	ja	24, 48	RGB	320	240	US	nein	ja	IMG043
30	FA32IMG384X288X24	ja	24, 48	RGB	384	288	Eu	nein	ja	IMG044
31	FA32IMG592X442X24US	ja	24, 48	RGB	592	442	US	ja	nein	IMG045
32	FA32IMG592X442X24	ja	24, 48	RGB	592	442	Eu	ja	nein	IMG046
33	FA32IMG640X480X24	ja	24, 48	RGB	640	480	US	nein	ja	IMG047
34	FA32IMG768X576X24	ja	24, 48	RGB	768	576	Eu	nein	ja	IMG048
	Color DIB									
35	FG32DIB160X120X24	grbflg	24 bit	RGB	160	120	US	ja	nein	IMG051
36	FG32DIB192X144X24	grbflg	24 bit	RGB	192	144	Eu	ja	nein	IMG052
37	FG32DIB320X240X24	grbflg	24 bit	RGB	320	240	US	ja	nein	IMG053
38	FG32DIB384X288X24	grbflg	24 bit	RGB	384	288	Eu	ja	nein	IMG054
39	FG32DIB592X442X24US	grbflg	24 bit	RGB	592	442	US	ja	nein	IMG055
40	FG32DIB592X442X24	grbflg	24 bit	RGB	592	442	Eu	ja	nein	IMG056
41	FG32DIB640X480X24	grbflg	24 bit	RGB	640	480	US	ja	nein	IMG057
42	FG32DIB768X576X24	grbflg	24 bit	RGB	768	576	Eu	ja	nein	IMG058
	Color DIB Online									
43	FG32DIB160X120X16IN24	ja, 15	ja	RGB	160	120	US	ja	nein	IMG061
44	FG32DIB192X144X16IN24	ja, 15	ja	RGB	192	144	Eu	ja	nein	IMG062
45	FG32DIB320X240X16IN24	ja, 15	ja	RGB	320	240	US	ja	nein	IMG063
46	FG32DIB384X288X16IN24	ja, 15	ja	RGB	384	288	Eu	ja	nein	IMG064
47	FG32DIB592X442X16IN24US	ja, 15	ja	RGB	592	442	US	ja	nein	IMG065
48	FG32DIB592X442X16IN24	ja, 15	ja	RGB	592	442	Eu	ja	nein	IMG066

49	FG32DIB640X480X16IN24	ja, 15	ja	RGB	640	480 US	ja	nein	IMG067
50	FG32DIB768X576X16IN24	ja, 15	ja	RGB	768	576 Eu	ja	nein	IMG068
	Color DIB 15								
51	FG32DIB160X120X16	ja, 15	ja	555	160	120 US	ja	nein	IMG071
52	FG32DIB192X144X16	ja, 15	ja	555	192	144 Eu	ja	nein	IMG072
53	FG32DIB320X240X16	ja, 15	ja	555	320	240 US	ja	nein	IMG073
54	FG32DIB384X288X16	ja, 15	ja	555	384	288 Eu	ja	nein	IMG074
55	FG32DIB592X442X16US	ja, 15	ja	555	592	442 US	ja	nein	IMG075
56	FG32DIB592X442X16	ja, 15	ja	555	592	442 Eu	ja	nein	IMG076
57	FG32DIB640X480X16	ja, 15	ja	555	640	480 US	ja	nein	IMG077
58	FG32DIB768X576X16	ja, 15	ja	555	768	576 Eu	ja	nein	IMG078
	Color 15								
59	FG32IMG160X120X16	grbflg	24, 48	555	160	120 US	nein	nein	IMG081
60	FG32IMG192X144X16	grbflg	24, 48	555	192	144 Eu	nein	nein	IMG082
61	FG32IMG320X240X16	grbflg	24, 48	555	320	240 US	nein	nein	IMG083
62	FG32IMG384X288X16	grbflg	24, 48	555	384	288 Eu	nein	nein	IMG084
63	FA32IMG592X442X16US	grbflg	24, 48	555	592	442 US	nein	nein	IMG085
64	FA32IMG592X442X16	grbflg	24, 48	555	592	442 Eu	nein	nein	IMG086
65	FG32IMG640X480X16	grbflg	24, 48	555	640	480 US	nein	nein	IMG087
66	FG32IMG768X576X16	grbflg	24, 48	555	768	576 Eu	nein	nein	IMG088
	Color 16								
67	FG32IMG160X120X16AS565	grbflg	24, 48	565	160	120 US	nein	nein	IMG091
68	FG32IMG192X144X16AS565	grbflg	24, 48	565	192	144 Eu	nein	nein	IMG092
69	FG32IMG320X240X16AS565	grbflg	24, 48	565	320	240 US	nein	nein	IMG093
70	FG32IMG384X288X16AS565	grbflg	24, 48	565	384	288 Eu	nein	nein	IMG094
71	FA32IMG592X442X16AS565US	grbflg	24, 48	565	592	442 US	nein	nein	IMG095
72	FA32IMG592X442X16AS565	grbflg	24, 48	565	592	442 Eu	nein	nein	IMG096
73	FG32IMG640X480X16AS565	grbflg	24, 48	565	640	480 US	nein	nein	IMG097
74	FG32IMG768X576X16AS565	grbflg	24, 48	565	768	576 Eu	nein	nein	IMG098
75	FG32IMGXXX	nein	32	32 bit	dwords	- -	nein	nein	FUN009
76	FG32DIBXXX	nein	32	32 bit	xxx	yyy -	ja	nein	FUN008
100	DDRSW08	nein	8	8	dx	zcount	zoom =	1	FKT090
101	DDRSW15	nein	8	555	dx	zcount	zoom =	1	FKT091
102	DDRSW16	nein	8	565	dx	zcount	zoom =	1	FKT092
103	DDRSW24	nein	8	888	dx	zcount	zoom =	1	FKT093
104	DDRSW32	nein	8	0888	dx	zcount	zoom =	1	FKT094
105	DDRCO08	nein	16	8	dx	zcount	zoom =	1	FKT095
106	DDRCO15	nein	16	555	dx	zcount	zoom =	1	FKT096
107	DDRCO16	nein	16	565	dx	zcount	zoom =	1	FKT097
108	DDRCO24	nein	16	888	dx	zcount	zoom =	1	FKT098
109	DDRCO32	nein	16	0888	dx	zcount	zoom =	1	FKT099
110	DDR2SW08	nein	8	8	dx	zcount	zoom =	2	FKT100
111	DDR2SW15	nein	8	555	dx	zcount	zoom =	2	FKT101
112	DDR2SW16	nein	8	565	dx	zcount	zoom =	2	FKT102
113	DDR2SW24	nein	8	888	dx	zcount	zoom =	2	FKT103
114	DDR2SW32	nein	8	0888	dx	zcount	zoom =	2	FKT104
115	DDR2CO08	nein	16	8	dx	zcount	zoom =	2	FKT105
116	DDR2CO15	nein	16	555	dx	zcount	zoom =	2	FKT106
117	DDR2CO16	nein	16	565	dx	zcount	zoom =	2	FKT107
118	DDR2CO24	nein	16	888	dx	zcount	zoom =	2	FKT108
119	DDR2CO32	nein	16	0888	dx	zcount	zoom =	2	FKT109
120	DDR4SW08	nein	8	8	dx	zcount	zoom =	4	FKT110
121	DDR4SW15	nein	8	555	dx	zcount	zoom =	4	FKT111
122	DDR4SW16	nein	8	565	dx	zcount	zoom =	4	FKT112
123	DDR4SW24	nein	8	888	dx	zcount	zoom =	4	FKT113
124	DDR4SW32	nein	8	0888	dx	zcount	zoom =	4	FKT114
125	DDR4CO08	nein	16	8	dx	zcount	zoom =	4	FKT115
126	DDR4CO15	nein	16	555	dx	zcount	zoom =	4	FKT116
127	DDR4CO16	nein	16	565	dx	zcount	zoom =	4	FKT117
128	DDR4CO24	nein	16	888	dx	zcount	zoom =	4	FKT118
129	DDR4CO32	nein	16	0888	dx	zcount	zoom =	4	FKT119

The column WinNT Fnr contains IoCtl codes, which are defined in the header file FgIoctl.h.

All these functions must be called with a pointer to the following data structure DIRECTXPARAMS:

```
typedef struct
    {
    PBYTE ptr;              // Adresse TopLeft
    ULONG dx;               // Breite in Pixeln
    ULONG zcount;           // Anzahl der Zeilen
    ULONG zoffset;          // Zeilenoffset in Bytes
    ULONG zlen;             // Zeilenlänge in Bytes
    ULONG av                // average anzahl
    ULONG basis;            // basisadresse for direct x
    ULONG reserved[8]
    } DIRECTXPARAMS;
```

For functions 1-99 it is sufficient to provide a pointer and the base address of the Frame Grabber card.

Functions 100-129 (DDR stands for Direct Draw) have to be called twice for interlaced mode images.
Zoffset will be added to zlen, in order to skip every second line. This is shown in Low-Level source code examples for interlaced mode formats. Between odd and even fields some blind reads are

required. Based on imode*zlen bytes, these blind reads allow for both fields to be adjusted to the right place.

There are functions that show 2x and 4x zoomed images. In this case zlen or 3*zlen must be adjusted to skip the right number of VGA lines. All these DirectX functions can be used to write directly into DIBs as well. Zoffset can also be zero or have negative values.

1.3 Low-Level Programming under Windows Me/ 98/ 95/ 3.x/ DOS

1.4 Low-Level Programming under DOS

Under DOS, Windows 3.0, Windows 3.1, Windows 3.11, Windows 95, Windows 98 and Windows Me Low-Level calls are made by software interrupt 60H using registers ax, bx, cx and dx. In section 5, a detailed description is followed by compiler specific-notes that explain how to implement Interrupt 60H calls.

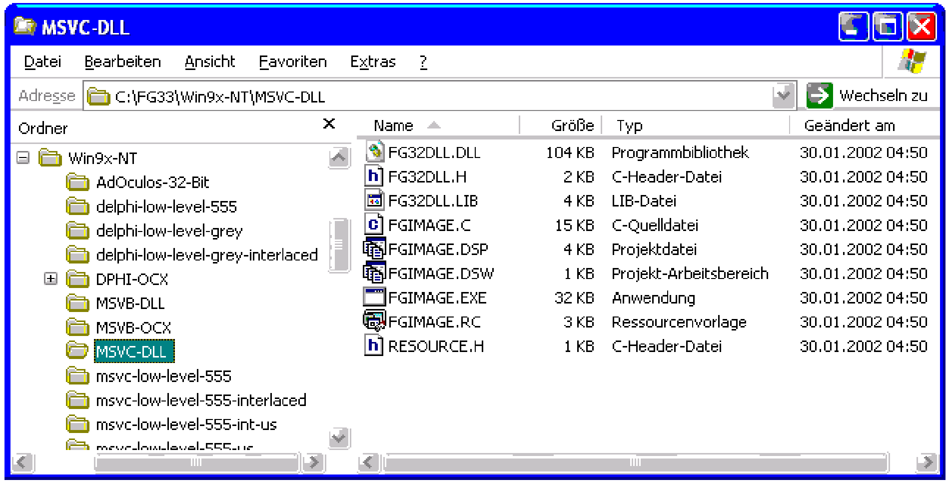
1.5 Low-Level Programming under OS/2

Under OS/2 the device driver call is made by a function with the name DosDevIOCtl, which is similar to the one described above in section 1.2. Details are given in chapter 8. There you can find OS/2 source-code examples for Borland C and IBM C/2.
For this operating system only a part of the shown data transfer functions exist. These functions are described in the programming examples. Under OS/2 the IOPL level can be switched to IOPL=2. In this case user programs have direct access to I/O ports.

II.
32-Bit Programming on a Procedural Level with MS-Windows 9x/ Me and MS-Windows XP/ 2000/ NT

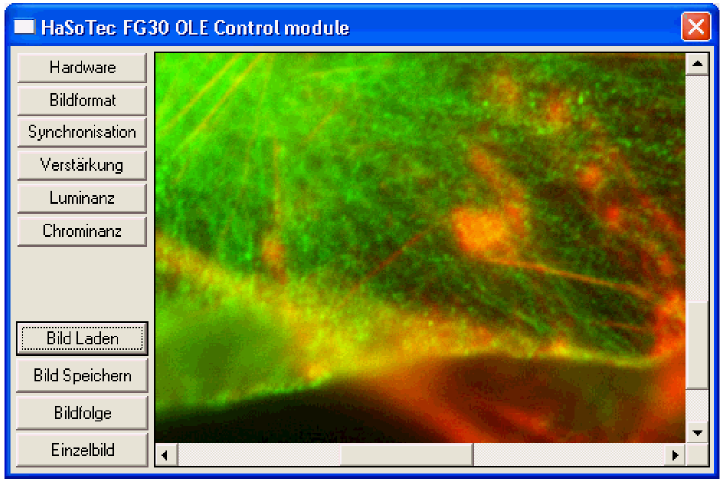
2.1 Programming under Microsoft Visual C++ 2.0, 4.0, 4.1, 4.2, 5.0 and 6.0 without OCX Control

Subdirectory ....\Win9x-NT\MSVC-DLL
contains the following files:

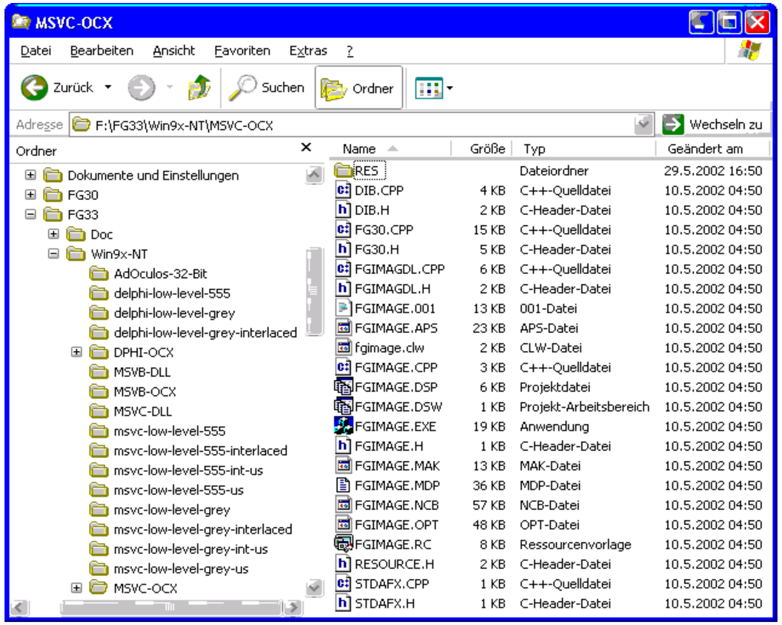


A description of how to use the program is contained in chapter 6. The file FG32DLL.DLL is required to run the program. There is a separate dialog and grab function for both color and grey images. A single dialog shows live display of the video source and has subdialogs to control nearly all adjustments of the cards. This example has the same functions as some 16-bit examples.

2.2 Programming under Microsoft Visual C++ 6.0, 5.0 and 4.2 with OCX Control



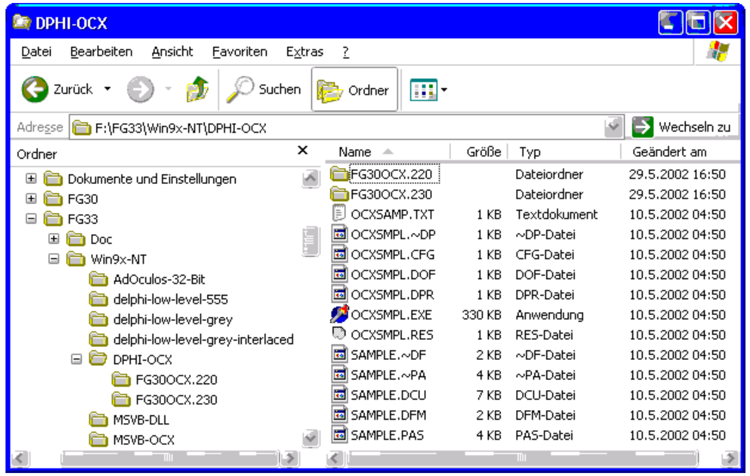
OCX controls can be integrated into modern compiler environments. Their functionality is described in chapter 9 in detail. In Microsoft Visual C++ there are wizards to implement controls. One can install a generated and modified example such as the following:

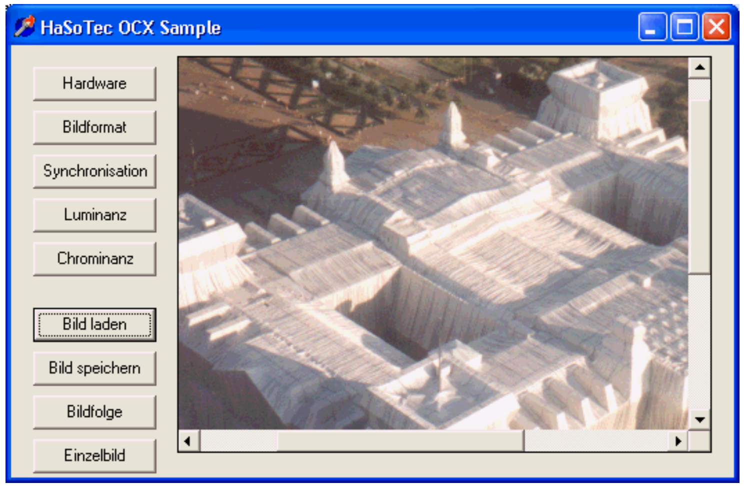


2.3 Programming under Borland Delphi 6.0 5.0, 4.0, 3.00, 3.01, 3.02 and 2.0 without OCX Control

As is the case with other Microsoft DLLs, DLLs made for Visual C++ and Visual Basic are unfortunately not compatible with Delphi. Low-Level examples for all formats are provided and can be used instead.

2.4 Programming under Borland Delphi 2.0 - 6.0 with OCX Control





Under Delphi the OCX controls must be imported, so the compiler automatically generates libraries, which contain jump tables to the functions of the control. Older Delphi versions may not be compatible with the current controls. You can find older controls in subdirectories and exchange them.

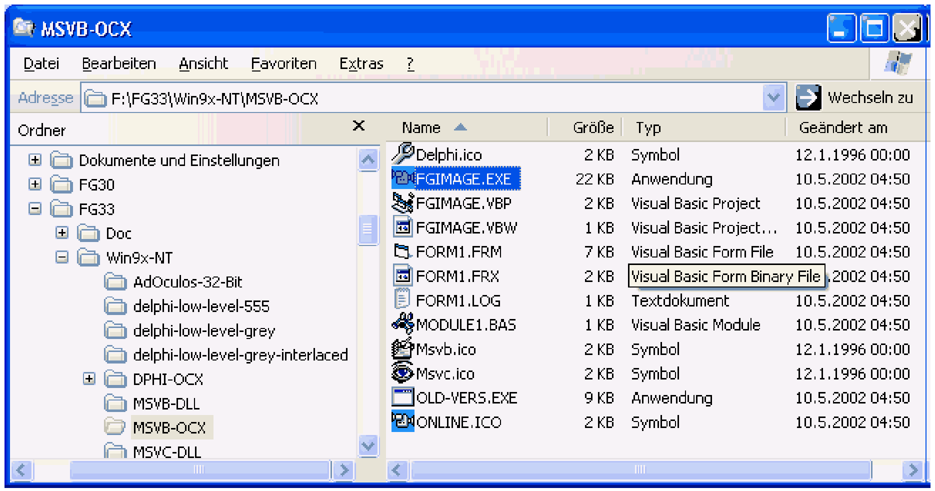
2.5 Programming under Borland C-Builder with OCX Control

The embedding of controls under C++ Builder and Delphi is practically identical.
No separate source code example is provided in versions before 4.83. Since this compiler can use mixed C and Pascal procedures, it should be possible to start with the Delphi example.

2.6 Programming under Borland C++ 5.01 without OCX Control

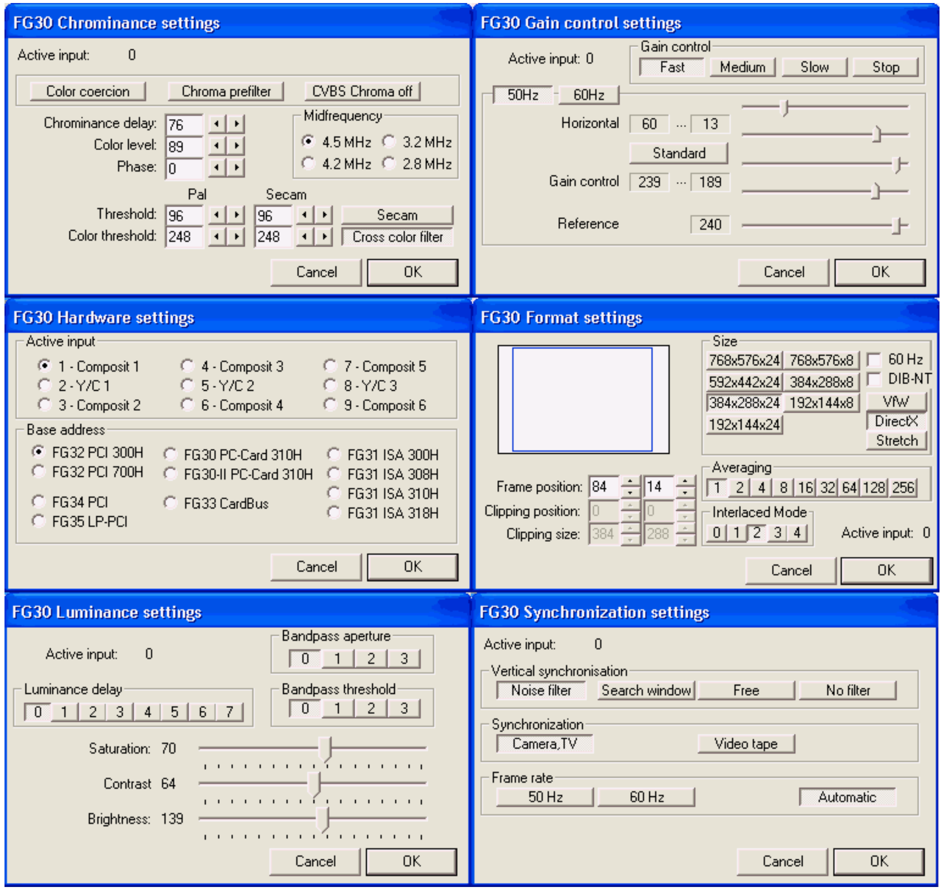
Microsoft DLLs are not compatible with this compiler. Low-Level C-Builder examples (Version 4.81 or later) can be used instead.

2.7 Programming under Visual Basic 6.0, 5.0 and 4.0 with OCX Control



The example is similar to the Microsoft C++ example.

Dialogs are similar in all OCX examples and are shown below:



III.
Programming on a Procedural Level with MS-Windows 3.x and MS-Windows 9x

In this chapter the term "procedure level" means that ready-to-use functions implemented in libraries are employed. No knowledge of device driver functions is required to use them. Functions which are included in a library are typically often-needed complex procedures. These functions contain MS-Windows API calls and Low-Level calls to FG30DRV.EXE.
If library functions are used in an application, this does not exclude the use of Low-Level functions in other parts of the same program.
Images under MS-Windows follow the standard of Device Independent Bitmaps (DIB). This small additional programming effort is efficient, because it enables MS-Windows to display images on any graphics board.
Several examples are supplied with the resource script files used by the libraries. In this case the programmer can define which dialog box elements are shown where, and it is easy to reduce the number of dialog box elements.

3.1. Programming in C

C is the best choice for writing applications under MS-Windows. The programming tools supplied with such compilers have the best quality and the number of sample applications is much higher than in other compiler products.
Unfortunately, the structure of libraries is not the same among the various compiler manufacturers. To avoid problems, all examples contain their own library.

3.1.1 Microsoft Visual C++ 1.0-1.52
3.1.2. Microsoft C/C++ 7.0

The folder WINMSC70 contains the following files::

```
  Name          Original    Packed  Ratio   Date      Time    Attr CRC
--------------  --------  -------- ------ -------- -------- ---- ----
  FG32IMG.CFG         32        23  76.7% 93-07-01 02:00:00 a--w D04E
  FGIMAGE            512       235  45.9% 93-03-15 01:01:00 a--w 7C06
  FGIMAGE.C        13056      3866  29.6% 93-03-15 01:01:00 a--w 6A64
  FGIMAGE.DEF        256       171  66.8% 93-03-15 01:01:00 a--w B029
  FGIMAGE.DLG      16896      3589  21.2% 93-07-01 02:00:00 a--w 4AA3
  FGIMAGE.EXE      49104     21122  43.0% 93-07-01 02:00:00 a--w 875A
  FGIMAGE.H         4096      1022  25.0% 93-07-01 02:00:00 a--w 9C3F
  FGIMAGE.ICO        766       169  22.1% 93-03-15 01:01:00 a--w 0492
  FGIMAGE.LIB      54303     23318  42.9% 93-07-01 02:00:00 a--w 3E1C
  FGIMAGE.RC         512       220  43.0% 93-03-15 01:01:00 a--w BDFF
  README.TXT         363       232  63.9% 93-07-01 02:00:00 a--w E647
--------------  --------  -------- ------ -------- --------
    11 files      139894     53967  38.6% 93-07-01 02:00:00
```

Time and date of a file show the version numbers. To make updates easy, it is recommended that the user change only the following files:

FGIMAGE.C	-	source code example
FGIMAGE.DEF	-	modul definition file
FGIMAGE.ICO	-	Icon
FGIMAGE.RC	-	Resource script file
FGIMAGE.EXE	-	compiled example
FGIMAGE.CFG	-	configurations file (FG-30)
FGIMAGE.	-	Make / Nmake file for MS-C

The following files are supplied by HaSoTec with updates:

FGIMAGE.DLG	-	Resource scripts of dialogs contained in libraries
FGIMAGE.H	-	data structures and index definitions
FGIMAGE.LIB	-	library

The library FGIMAGE.LIB contains functions which follow the user documentation of chapter 6 (Ad Oculos). The appearance of dialog boxes on the screen and the user interface is equivalent to that of the Ad Oculos driver.

int FAR PASCAL LibMain (*hwnd*, *r0, r1, lpstr*)

HWND *hwnd*	-	Client Window handle
WORD *r0*	-	reserved = 0
WORD *r1*	-	reserved = 0
LPSTR *lpstr*	-	reserved = NULL

LibMain initializes the library and frame grabber by reading a configuration file. If no configuration data is present, a default data set is used.

returns	-	0 for success

int FAR PASCAL SNAPDLG (*hwnd, msg, wpar, lpar*)

HWND *hwnd*	-	Client Window handle
unsigned *msg*	-	Message parameter as is used in any dialog box procedure
WORD *wpar*	-	Word parameter
LONG *lpar*	-	Long parameter
returns	-	0 for success

SNAPDLG ist a dialog box function for capturing grey-level images. The image is shown in a dialog box with various control buttons.

The buttons allow you to open 5 additional dialog boxes for making

a variety of adjustments to the FG-3x board.

It is possible to freeze the image, to select averaging 1,2,4,6,16... and to select a frame rate. This function operates with all graphics boards under Microsoft Windows, but it is advisable to use a board with at least 256 colors so as to ensure quality grey-level images.

It is possible to change between 50 Hz and 60 Hz TV standards. One can select the following image resolutions:

50 Hz	60 Hz
768x576	640x480
384x288	320x240
192x144	160x120

All pixels in all formats and standards are square pixels. Low-Level functions can be used to select a hardware window based on the shown resolutions to speed up access times to image segments.

int FAR PASCAL TakeFg32Image (*phin, phout, pfgi, pfgo, hwnd*)

HANDLE FAR * *phin*

Pointer to an array of handles of input images (is 0 in this case of not having any input images)

HANDLE FAR * *phout*

Pointer to an array of handles of output images (this array may contain only 1 dimension for this case)

FGINFO FAR * *pfgi* — Pointer to a FGINFO structure (may be set to 0)

FGINFO FAR * *pfgo*

	Pointer to a FGINFO structure for output images
HWND *hwnd*	Client window handle.
returns	error code or 0

TakeFG32Image can fill image data to a buffer in memory. If an image was frozen in the dialog box procedure SNAPDLG, the last frozen image is taken by this function. In all other cases the actual camera image is taken. All source code examples show how to fill image data into a device-independent bitmap (DIB).

int FAR PASCAL SNAPDLGRGB (*hwnd, msg, wpar, lpar*)

HWND hwnd	-	Client Window handle
unsigned msg	-	Message parameter as is used in dialog box procedures
WORD wpar	-	Word parameter (dialog box procedure)
LONG lpar	-	Long parameter (dialog box procedure)
returns	-	0 for success

SNAPDLGRGB ist a dialog box function for capturing true-color images. The image is shown in a dialog box with various control buttons.

The buttons allow you to open 5 additional dialog boxes for making a variety of adjustments to the FG-3x board.

It is possible to freeze the image and to select averaging 1,2,4,6,16... This function operates with all graphic boards under Microsoft Windows, but if you use an ET4000 HiColor board with a driver that supports 32768 colors, this will significantly increase the speed of the color display. In this case the graphic output is performed by a special device driver with approximately 8

frames/sec.

The same dialog boxes as in the function SNAPDLG can be selected with various buttons. The programmer need not worry about the adjustments in detail. A single function call shows a live imagine such that all adjustments can be made with the help of integrated dialogs..

int FAR PASCAL TakeFg30ImageRgb (*phin, phout, pfgi, pfgo, hwnd*)

HANDLE FAR * *phin*	Pointer to an array of handles of input images
HANDLE FAR * phout	Pointer to an array of handles of output images (this array contains 3 dimensions for each R, G and B part of the image)
FGINFO FAR * *pfgi*	Pointer to a FGINFO structure
FGINFO FAR * *pfgo*	Pointer to a FGINFO structure for output images
HWND *hwnd*	Client window handle.
returns	error code or 0

TakeFG30ImageRgb works like TakeFG30Image except that color images are captured instead of grey-level images.

VOID SizeWindow (*hwnd*)

HWND	hwnd	Client window handle

SizeWindow optimizes the window size for showing the last captured image. If an image is larger than a possible window on the screen, scroll bars are generated automatically. This function works correctly also for menu bars which need 2 lines of text.

BOOL MakeDialog (lpstr, hwnd, farproc)

LPSTR lpstr	Dialog names string
HWND hwnd	Client window handle
FARPROC farproc	Dialog box procedure name
returns	true for success

MakeDialog simplifies dialog calls.

3.1.3. Borland C++ 3.1, C++ 4.0, C++ 4.5

The subdirectory WINBRLND contains the following files:

Name	Original	Packed	Ratio	Date	Time	Attr	CRC
FG32IMG.CFG	30	25	83.3%	93-07-01	02:00:00	a--w	C1C3
FGIMAGE.C	12935	3875	30.0%	93-03-15	01:01:00	a--w	7FE5
FGIMAGE.DEF	236	163	69.1%	93-03-15	01:01:00	a--w	D760
FGIMAGE.DLG	16863	3587	21.3%	93-07-01	02:00:00	a--w	B00F
FGIMAGE.DSK	1472	570	38.7%	93-03-15	01:01:00	a--w	6053
FGIMAGE.H	4522	1288	28.5%	93-07-01	02:00:00	a--w	138E
FGIMAGE.ICO	766	169	22.1%	93-07-01	01:01:00	a--w	0492
FGIMAGE.LIB	60416	25954	43.0%	93-07-01	02:00:00	a--w	2900
FGIMAGE.PRJ	3856	988	25.6%	93-03-15	01:01:00	a--w	DF3B
FGIMAGE.RC	935	424	45.3%	93-03-15	01:01:00	a--w	7350
README.TXT	363	232	63.9%	93-07-01	02:00:00	a--w	E647
FGIMAGE.EXE	71919	26211	36.4%	93-07-01	02:00:00	a--w	EA9B
12 files	174313	63486	36.4%	93-07-01	02:00:00		

Time and date of a file show the version numbers. To make updates easy it is recommended that the user change only the following files:

FGIMAGE.C	-	source code example
FGIMAGE.DEF	-	modul definition file
FGIMAGE.ICO	-	icon
FGIMAGE.RC	-	resource script file
FGIMAGE.EXE	-	compiled result
FGIMAGE.CFG	-	configurations data
FGIMAGE.PRJ	-	project file
FGIMAGE.DSK	-	project file

The following files are supplied by HaSoTec with updates:

FGIMAGE.DLG	-	resource script file for library dialogs
FGIMAGE.H	-	data structures and index definitions
FGIMAGE.LIB	-	library

All functions are described in section 3.1.2.

3.2. Programming in C++

3.2.1. Microsoft Visual C++ 1.0 ... 1.52
3.2.2. Microsoft C/C++ 7.0

The WINMSCPP subdirectory contains the following files:

Name	Original	Packed	Ratio	Date	Time	Attr	CRC
FG32IMG.CFG	30	23	76.7%	93-07-01	02:00:00	a--w	D04E
FGIMAGE.CPP	6528	2104	32.2%	93-07-01	02:00:00	a--w	7A3E
FGIMAGE.DEF	256	155	60.5%	93-07-01	02:00:00	a--w	FA43
FGIMAGE.DLG	16896	3589	21.2%	93-07-01	02:00:00	a--w	4156
FGIMAGE.EXE	71168	32281	45.4%	93-07-01	02:00:00	a--w	997D
FGIMAGE.H	4608	1205	26.2%	93-07-01	02:00:00	a--w	2171
FGIMAGE.ICO	766	169	22.1%	93-03-15	01:01:00	a--w	0492
FGIMAGE.LIB	54303	23318	42.9%	93-07-01	02:00:00	a--w	3E1C
FGIMAGE.RC	2560	683	26.7%	93-07-01	02:00:00	a--w	795B
FGIMAGEC.C	7168	2061	28.8%	93-07-01	02:00:00	a--w	D1D8
MAKEFILE	640	289	45.2%	93-07-01	02:00:00	a--w	C6EF
11 files	164923	65877	39.9%	93-07-01	02:00:00		

For the description of all functions please refer to chapter 3.1.2. The files FGIMAGE.DLG, FGIMAGE.LIB, FGIMAGE.H are subject to changes in further updates.

3.3. Programming in Basic

3.3.1. Microsoft Visual Basic

The manipulation of image data is not very flexible under Visual Basic 1.0. The evolution of the programming language Quick Basic has shown that later versions contain the necessary tools to manipulate data arrays, so one can expect that future versions of Visual Basic will add new useful functions.
In Visual Basic 2.0, 3.0 or Visual Basic Professional, some such functions have already been introduced. The source code example has been tested with Version 1.0 and should run without modifications in later versions.

This program is very short because it makes use of the Visual Basic variable “picture”, which assigns a Bitmap to the client window. One can thus avoid all programming effort necessary for realizing a paint structure.
The programming example contains in its corresponding library FG30VB.DLL functions which give line-by-line access to image data contained in a Device Independent Bitmap (DIB). Powerful functions for manipulating data arrays are missing in current compiler versions. FG30VB.DLL makes at least slow manipulations of frame data possible. As an example, a function for inverting a grey-level image is included.

The subdirectory WINMSVB contains the following files:

Name	Original	Packed	Ratio	Date	Time	Attr	CRC
FG32IMG.CFG	30	24	80.0%	93-07-01	02:00:00	a--w	F27D
FG32VB.DLL	46720	20486	43.8%	93-07-01	02:00:00	a--w	D7EF
FGIMAGE.BAS	1527	426	27.9%	93-07-01	02:00:00	a--w	1086
FGIMAGE.EXE	11041	4297	38.9%	93-07-01	02:00:00	a--w	384B
FGIMAGE.FRM	6747	2479	36.7%	93-07-01	02:00:00	a--w	C423
FGIMAGE.MAK	63	61	96.8%	93-07-01	02:00:00	a--w	FD71
6 files	66128	27773	42.0%	93-07-01	02:00:00		

The functions used under Visual Basic can be described as follows:

Declare Function SNAPDIALOGS Lib "FG32VB.DLL" (ByVal hWnd, ByVal i As Integer)

hWnd - property of applications window
i - index of dialog to call

i=0: calls a dialog box for digitizing grey-level images. A slow display of the video source with selectable dialog boxes is available simultaneously. This function works on all graphics boards, but a board with at least 256 colors is recommended.

i=1: calls a dialog box function used to digitize true-color images. A continuous display of the video source is automatically activated if an ET4000 graphics board is used in 32,768 color mode. In all other cases only single frames can be captured.
HiColor resolutions are recommended.

Declare Function TakeFg32Img Lib "FG32VB.DLL" (hin, hout, lpInfIn As FGINFO, lpInfOut As FGINFO, ByVal hWnd)

hin points to a handle of an input image and is set to 0 in this version. Please note that the pointer function is realized if a ByVal statement is not used.
hout points to a handle of an output image. Its value is taken from an API call to GlobalAlloc.
lpInfIn points to a type declaration FGINFO for an input image and is set to 0.
lpInfOut points to a FGINFO type declaration for the output image.
hWnd is a property of the application window

TakeFG30Image takes either a frozen or an actual grey-level image, depending on the last dialog-box settings. Image data is taken into a linear frame buffer.

Declare Function TakeFg32ImgRgb Lib "FG32VB.DLL" (hin, hout, lpInfIn As FGINFO, lpInfOut As FGINFO, ByVal hWnd)

hin points to an array containing 3 handles to 3 input images. In this version this parameter can be set to 0. Please note that the pointer function is reached if the parameters are placed in the function call without a ByVal statement.
hout points to an array containing 3 handles for 3 output images. These values are the result of GlobalAlloc function calls to the MS-Windows API.
lpInfIn points to a type declaration FGINFO and can be set to 0.
lpInfOut points to a FGINFO type declaration to produce 3 output images containing red, green and blue image data.
hWnd is a property of the application window

TakeFG32ImageRgb takes either a frozen or a camera image, depending on the last settings made by the corresponding dialog box calls.

Declare Function GetBufLine Lib "FG32VB.DLL" (ByVal pbuf As Long, ByVal xsize, ByVal y, a As Integer)

pbuf points to a data buffer which can be of any size (no 64K limit). Such Data buffers can be allocated with a GlobalAlloc function call to the Windows API. The result of such a call is the pbuf parameter.

xsize is the number of bytes to be taken from the buffer. In the case of the source code examples, this is the length of one line of an image.

y is the cycle number of the transfer (line number)

a is a data array into which the required values are placed. This array should have at least xsize number of elements. Each element of this array is used to contain the value of a single byte of the data buffer.

The first byte of the buffer starts with line number y=0.

Declare Function GetDibLine Lib "FG32VB.DLL" (ByVal pdib As Long, ByVal y, a As Integer)

pdib points to a DIB of unlimited size (no 64K limit). This parameter can be produced by a GlobalAlloc/GlobalLock function call to the Windows API.

y is the line number of the image.

a is a data array of integer values, which has at least as many elements as pixels are possible per image line. Each element corresponds to a single byte of an image line.

This function reads a line of an image presented in a Device-Independent Bitmap (DIB) format. Before this function is called, the DIB must have a valid BITMAPINFOHEADER.
Please refer also to the MakeColorHeader and MakeGreyHeader functions.

Declare Function SetDibLine Lib "FG32VB.DLL" (ByVal pdib As Long, ByVal x, a As Integer)

pdib points to a DIB of any size. This parameter normally is produced by a GlobalAlloc/GlobalLock function call.

y is the line number of an image.

a points to a data array of integer values. This array is at least as large as pixels per line can be expected.

This function writes an image line directly into a Device-Independent Bitmap (DIB).
Before this function is called the DIB must have valid BITMAPINFOHEADER information as is produced by the functions MakeColorHeader and MakeGreyHeader.

Declare Function MakeGreyHeader Lib "FG32VB.DLL" (ByVal pdib As Long, ByVal x, ByVal y)

pdib points to a data buffer to be used as a DIB. Such data buffers can be allocated by using GlobalAlloc/GlobalLock function calls to the Windows API.

x is the number of pixels per line of the required DIB

y is the number of image lines of the required DIB.

This function generates a BITMAPINFOHEADER and a grey-level palette.

Declare Function MakeColorHeader Lib "FG32VB.DLL" (ByVal pdib As Long, ByVal x, ByVal y)

pdib points to a DIB. Such data buffers are created with GlobalAlloc and GlobalLock function calls

x is the number of pixels per line of an image to be placed in the DIB

y is the number of image lines

This function prepares a BITMAPINFOHEADER as is required by DIB, which contains true-color image data without palette information.

Declare Function SizeWindow Lib "FG32VB.DLL" (ByVal hWnd, ByVal hdib As Integer)

hWnd a property of the main application window.
hdib is a handle to a valid DIB.

This function adjusts the window size to the size of a DIB image.

Declare Function WriteBmpFromDib Lib "FG32VB.DLL" (ByVal dibptr As Long)

dibptr points to a buffer of a DIB.

This function writes an MS-Windows Bitmap to disk, using the name FGIMAGE.BMP. Such files can also be used by other applications such as Paintbrush.

3.4. Programming in Pascal

3.4.1. Borland Turbo Pascal for Windows 7.0

The subdirectory WINPAS70 contains the following files:

```
  Name          Original    Packed  Ratio   Date      Time    Type  CRC
--------------  --------  --------  ------ -------- -------- ----- ----
  FGIMAGE.PAS       8934      2365  26.5% 93-07-01 02:00:00 -lh5- BE7B
  FGTPWLIB.DLL     47296     20755  43.9% 93-07-01 02:00:00 -lh5- FB78
  FGIMAGE.EXE      58900     27768  47.1% 93-07-01 02:00:00 -lh5- C5C3
  FGIMAGE.RES        178       108  60.7% 93-07-01 02:00:00 -lh5- 7361
  FG32IMG.CFG         30        25  83.3% 93-07-01 02:00:00 -lh5- 8F45
--------------  --------  --------  ------ -------- --------
     5 files      115338     51021  44.2% 93-07-01 02:00:00
```

function SnapDialogs (Window: HWnd; i: Word):Word;**far**;
external 'FGTPWLIB' **name** 'SNAPDIALOGS';

hWnd Handle of applications window
i index to the dialog to be called

i=0: is a dialog-box function for digitizing grey-level images. A display of the video source is shown and access to several dialogs is available.
To display good quality grey-level images, a graphic board with at least 256 colors is recommended.
i=1: is a dialog box function for digitizing true-color images. On ET4000 HiColor boards a video source can be displayed with high frame rates.

function TakeFg32Img (lphin: LPHandle; lphout:
LPHandle; fg: LPFGINFO; fg: LPFGINFO;
Window: HWND):Word;**far**; **external**
'FGTPWLIB' **name** 'TAKEFG32IMG';

lphin points to a handle of an input image. In this version

	this function has no input images, so the value can therefore be set to 0.
lphout	points to a handle of an output image. Its value is returned by a previous GlobalLock call.
pfgi	points to an FGINFO data structure and can be set to 0
pfgo	points to a FGINFO data structure to describe an output image.
Window	Handle of the applications window

TakeFG30Image provides a frozen or an actual image (depending on dialog box settings). Data is taken into a linear frame buffer in memory.

function TakeFg32ImgRgb (lphin: LPHandle;
lphout: LPHandle; fg: LPFGINFO; fg: LPFGINFO;
Window: HWND):Word;**far**; **external**
'FGTPWLIB' **name** 'TAKEFG32IMGRGB';

hin	points to a data array containing 3 handles of 3 input images. Until this information is used in future versions, this value can be set to 0.
hout	points to a data array containing 3 handles for output images of the red, green and blue parts of the image. These handles can be allocated using the Windows API function GlobalAlloc.
lpInfIn	points to an FGINFO data structure defined for 3 input images (can be set to 0)
lpInfOut	points to a FGINFO data structure defined for 3 output images (red, green and blue)
hWnd	Handle of applications window

TakeFG30ImageRgb provides a frozen or actual true-color image (depending on the last dialog box settings)

Image data are taken into 3 frame buffers for red, green and blue parts of the image separately.

function MakeGreyHeader (lpdib: LPDIBINFO; x: Word;
y: Word):Word;**far**; **external** 'FGTPWLIB'
name 'MAKEGREYHEADER';

lpdib	points to a DIB. A DIB buffer can be allocated with the help of a GlobalAlloc function call to the MS-Windows API. The function GlobalLock returns the required pointer.
x	shows the number of pixels per line of the DIB image.
y	shows the number of lines per frame to be contained in the DIB.

This function generates a BITMAPINFOHEADER and a grey-scale palette

function MakeColorHeader (lpdib:LPDIBINFO; x: Word;
y: Word):Word;**far**; **external** 'FGTPWLIB'
name 'MAKECOLORHEADER';

lpdib	points to a buffer for a DIB. Such buffers can be allocated with GlobalAlloc/GlobalLock.
x	shows the number of pixels per line to be placed into the DIB
y	shows the number of lines of an image to be placed into the DIB.

This function generates a BITMAPINFOHEADER for a true-color DIB without a palette.

function PlaceDibBits (buf: PCHAR; dib: PCHAR ;

```
x: Word; y: Word):Word;far; external 'FGTPWLIB'
                          name 'PLACEDIBBITS';
```

buf	pointer to a buffer with linear image data
dib	Pointer to a DIB
x	number of pixels per line
y	number of lines per image

This function copies image data from a linear buffer into a DIB.

function PlaceDibBitsRgb (bufr: PCHAR; bufg: PCHAR;
bufb: PCHAR; dib: PCHAR ; x: Word;
y: Word):Word;**far**; **external** 'FGTPWLIB'
name 'PLACEDIBBITSRGB';

bufr	pointer to a buffer with red image data
bufg	pointer to a buffer with green image data
bufb	pointer to a buffer with blue image data
dib	pointer to a DIB
x	number of pixels per line
y	number of lines per frame

This function copies true-color data from 3 buffers into a DIB.

function SizeWindow (Window: HWND; lpbi:
LPDIBINFO):Word;**far**; **external** 'FGTPWLIB'
name 'SIZEWINDOW';

Window	handle to applications window
lpbi	handle to a valid DIB

This function adjusts the size of the applications window according to the image size of a DIB.

3.4.2. Borland Delphi 16-Bit

Under Borland Delphi 16-Bit the programmer has a simple tool to control Frame Grabber FG-30. FG30KOMP has grabbing and displaying functions implemented. A user interface to a linear frame buffer is provided to add user-specific functions inside the object-oriented compiler environment. TFG32Bitmap is an object that is compatible with the standard Borland object Tbitmap, so all Borland-provided functions can thus be used with it. The FG32KOMP component is based on the library FGTPWLIB.DLL and is the interface to the device driver Fg3xdrv. The executable files provided are built with Borland Delphi 1.0. Higher versions, which still have the 16-bit Compiler, can be used as well. This example cannot be used the 32-bit compiler environment. Please refer to section 6 of this chapter for OCX control examples or Low-Level examples.

To install the component, the following steps are required:

1. Import the FG30KOMP component into the palette of components
2. Install the Help file
3. Install the source code example

Start Installation"setup.exe" from the DINSTALL subdirectory and follow the steps in the setup. You must remember the chosen directories so that you can later fill them in manually in the Delphi environment. Normally all components can be found in the subdirectory \\Delphi\Lib. It is possible to use different subdirectories, but components and VCL cannot exceed 128 characters.

To import FG30KOMP, you must start Delphi and use the menu command "...install component". This opens a dialog box and "Insert" will ask for a complete path name of the component. You must switch the dialog box to *.dcu files until FG30KOMP.DCU can be selected. The compiler links the new file to its internal structures (COMPLIB) and after this process a component "FGIMAGE" is selectable.

At this time the library FGTPWLIB.DLL must be present in the component's directory or in \\Windows\System. You must watch for the availability of FGTPWLIB.DLL during compiler sessions and when running executable results. It makes sense to have the configuration file FG30IMG.CFG in the same directory, but if the program does not find it, it generates a new file with defaults.
To install Help please use the program "HelpInst" supplied by Delphi. When this program is started, either the file delphi.hdx or your own specific help file should be selected. It is normally found in the "\\delphi\Bin" subdirectory. All installed keyword files should now be displayed. A new file "FG32KOMP.KWF" can be attached, if "insert keyword file" is selected. Please note that the original help file (*.hlp) must be accessible for Delphi. Copy the file fg32komp.hlp to the \\Delphi\Bin subdirectory.

FG32Image is a container which has a drawing surface and an image data area IMAGE of the type TFG32Bitmap. It is compatible with the object type TBitmap. FG32IMAGE is not completely compatible with the TIMAGE component for several reasons, so it is therefore not safe to use TIMAGE-based functions.

Help functions of the unit:

IncW (p:pointer, tooadd:word)	helper function to increment pointers not limited to 64K segment size

Data types of TFG32Image

name	range of use	description
Imgkinds	(NONE, TRUECOLOR, GREYSCALE, PALCOLOR)	shows the type of the image for the Bitmap component
Badkinds	(H300,H308,H310, H318,H320,H328, u.s.w. bis H378)	This is a list of possible resources for the Frame Grabber card used
Anskinds	(red_cinch_connector, black_white_cinch connector_Hosiden_connector)	A list to be declared to switch betweend frame grabber inputs
TFG32Bitmap	pls refer to TFG32Bitmap	TBitmap-compatible class for dealing with bitmap-oriented images. There are many useful functions built into Delphi that can be used in conjunction with the grabbed images.
TFG32Filterevent	procedure (Sender: TObject;Filterindex :word) of object;	Type of onFilter event

properties of TFG32Image

Name of property	Type	at run time	description
autosize	Boolean		The size of the drawing area should automatically be adjusted to the image size
stretch	Boolean		Image display can be stretched to fit into the current window size
grabbonly	Boolean		Grabs frame into the linear frame buffer without display
backpal	Boolean		Realizes palette as a background palette
filterindex	Byte		If a filter is used, this value can act as a switch between filters
FG30Eingang	AnsKinds		Describes the FG-3x input of the frame grabber
Fg30Basisad	BadKinds		Base address of frame grabber
Image	TFG32Bitmap	x	TBITMAP compatible image object
ImgTypes	ImgKinds	x	Shows the type of currently stored frame in image buffer
MulipleFg30	Boolean		True, if more than one frame grabber should be controlled

Methods of TFG32Image

Name	Function
Greyimagedialog	Shows Dialog for making adjustments to grey-level images for the frame grabber

Colorimagedialog	Shows a dialog for making adjustments to the frame grabber used to grab color images
Loadbitmap(filename: String)	Loads a Windows Bitmap from a file. If the bitmap is in one of the savable formats, it is taken into the linear frame buffer
GrabbColorImage	Grabs and transfers a color frame from frame grabber
GrabbGreyImage	Grabs and transfers a grey frame from frame grabber
Showabout	About Dialogbox

Results of TFG32Image

Name	Function
onFilter	This event occurs every time grabbed image data are filled into the structure and are ready to be displayed. Frames are still not displayed and can be manipulated. TFG32Filterevent defines a Byte value that can be used as a filter index. As shown in the example, the hierarchy of this event is higher than that of the TFG32Image in order to ensure that the filter is processed first.

Properties of TFG32Bitmap, readable at run time

Name of property	Type	Description
fgstruc	array[0..2] of FGRec; FGRec = Record res :Word; {Reserved} breite :Word; Hoehe:word; end;	Information structure to keep information of the last digitized image

linbuf	Linbuf: array[0..2] of THandle;	Linear frame buffer, starting line-by-line from top left corner. The order of each Pixel is R-G-B (Bytes 0-1-2.)
size	Longint;	Determines the size of a color channel in Bytes

IV. Programming on the procedural level under DOS

4.1. Programming in C

4.2. Microsoft C/C++ 7.0

The subdirectory DOSMSC70 contains the following files:

```
  Name          Original    Packed  Ratio   Date     Time    Attr  CRC
--------------  --------  -------- ------ -------- -------- ---- ----
  DOSFG32.CFG        299        40  13.4% 93-07-01 02:00:00 a--w  6530
  DOSMSC70.C       21632      3261  15.1% 93-07-01 02:00:00 a--w  2759
  DOSMSC.EXE       47612     23375  49.1% 93-07-01 02:00:00 a--w  AD4E
  DOSMSC70.LIB     18332      7957  43.1% 93-07-01 02:00:00 a--w  88BD
  DOSMSC70.MAK       256        83  32.4% 93-07-01 02:00:00 a--w  61DF
--------------  --------  -------- ------ -------- --------
     5 files       88246     34716  39.3% 93-07-01 02:00:00
```

The library DOSMSC70.LIB contains the following functions:

void far pascal LIBMAIN (void)

This function initializes the FG-3x library. If the actual directory contains the file DOSFG3x.CFG the actual settings from this file are taken to initialize the library. If such a file was not found, the default values contained in the device driver FG3xDRV are valid.

void far pascal SETUP (void)

A setup menu similar to the program FG3xVGA presents itself. Use the [arrow keys] to reach an item and the [page up] and [page down] buttons value to modify the value. With the [Esc]ape key you leave the menu, and the configuration file DOSFG3x.CFG is updated to disk.

void far pascal GREY320 (pbuf)

char far * pbuf - pointer to a buffer with 320x240 Bytes

This function is used to digitize a grey-level image with 320x240 pixels in US standard.

void far pascal GREY384 (pbuf)

char far * pbuf - pointer to a buffer with 384x288 Bytes

This function is used to digitize grey-level images with 384x288 Pixels from video sources which operate with 50 Hz-TV standards.

void far pascal GREY640 (pbuf)

char far * pbuf - pointer to a buffer with 640x480 Bytes

This function is used to digitize a grey-level image with 640x480 pixels in US standard.

void far pascal GREY768 (pbuf)

char far * pbuf - pointer to a frame buffer containing 768x576 Bytes

This function is used to digitize grey level images with 768x576 Pixels from video sources which operate with 50 Hz-TV standards.

void far pascal GREY320AV (pbuf, av)

char far * pbuf - pointer to a frame buffer with 320x240x2 Bytes
int av - 2av= number of frames to average

This function is used to capture 320x240 pixel grey-scale images in US-standard with averaging.

void far pascal GREY384AV (pbuf , av)

char far * pbuf - pointer to a frame buffer with 384x288x2 Bytes
int av - 2av= number of frames to average

This function is used to capture 384x288 pixel grey-scale images in 50 Hz-standard with averaging.

void far pascal GREY640AV (pbuf, av)

char far * pbuf - pointer to a buffer 640x480 + 65536 Bytes
int av - 2av= number of frames to average

This function is used to digitize grey-scale images with 640x480 pixels from US-standard video sources with averaging. Because this function would need 640x480x2 Bytes, which are normally not available under DOS, 640 Kbytes are assigned to a temporary file on the current drive. This function is slow, but it works without DOS extenders or EMS drivers.
If XMS or EMS memory is available, it is advisable to rewrite this function to get higher performance or at least to use a RAM-drive as the current drive.

void far pascal GREY768AV (pbuf, av)

char far * pbuf - pointer to a frame buffer containing 768x576 + 65536 Bytes
int av - 2av= number of frames to average.

This function is used to digitize grey-scale images with 768x576

pixels from 50 Hz-standard video sources with averaging. Because this function would need 768x576x2 Bytes, which are normally not available under DOS, 896 Kbytes are assigned to a temporary file on the current drive. This function is slow, but it works without DOS extenders or EMS drivers.
If XMS or EMS memory is available, it is advisable to rewrite this function to get higher performance or at least to use a RAM-drive as the current drive.

void far pascal DISPGREY (pbuf, bits, xvga, yvga, ximg, yimg, xpos, ypos, xlen, ylen)

char far * pbuf	-	pointer to image buffer
int bits	-	Pixel depth of current VGA mode
int xvga	-	VGA x -resolution
int yvga	-	VGA y -resolution
int ximg	-	image data x -resolution
int yimg	-	image data y -resolution
int xpos	-	VGA x -position
int ypos	-	VGA y -position
int xlen	-	displayed x -resolution
int ylen	-	displayed y -resolution

This function is used to display grey-level images for test purposes. 4- und 8-bit modes of VGA boards are supported. The 4-bit-modes use 16 grey levels with resolutions of up to 640x480 pixels. On some SVGA boards this function works without problems for resolutions of 800x600 pixels. The 8-bit-mode supports the VGA resolution of 320x200 pixels. VGA boards with Tseng Labs ET4000 controller and some SVGA boards with similar memory page switching functions can work at up to resolutions of 800x600 and 1024x768. The VGA (x, y) position should be set to (0,0) for working with higher resolutions. In cases where that the image is larger than the VGA resolution, the image

can be cropped with the help of the parameters xlen, ylen.

int far pascal CHECKTSENG (void)

This function returns 0 if the graphic board is a ET4000 board.

void far pascal SWITCHTSENG (void)

This function switches a Tseng Labs ET4000 board into video mode 30H. This mode displays 800x600 pixels with 256 colors/grey levels.
Before using this function you should verify that the monitor connected to the graphic board can operate at this resolution.

void far pascal PALGREY16 (void)

A VGA-palette with 16 grey levels is applied.

void far pascal PALGREY256 (void)

A VGA-palette with 256 grey levels is applied.

void far pascal SETIMODE (imode)

int imode - interlaced mode 0 oder 1

Switches between 2 modes of recognizing even and odd fields in a video signal. This setup only has an effect at higher FG-30 resolutions.
The correct value should be tested for each video source. It is correct if the two half frames do not exhibit shifts in every second line – at least for non-moving objects.

void far pascal COLO320 (pbuf);

char far * pbuf - pointer to an image buffer with 320x240x3 Bytes

This function is used to capture true-color images with resolutions of 320x240x 24-bit from video sources working in US-TV-standard.

void far pascal COLO384 (pbuf)

char far * pbuf - a pointer to an image buffer with 384x288x3 Bytes

This function is used to capture true-color images with resolutions of 384x288x 24-bit from video sources working in 50 Hz-TV-standards.

void far pascal COLO640P1 (pbuf)

char far * pbuf - a pointer to an image buffer with 640x120x3 Bytes

This function is used to capture true color images with resolutions of 640x480x24-bit from video sources working in US-TV-standards.
Because of the limited memory for DOS applications, an image is digitized with the full size of 640x480 pixels, but only 1/4 of the image data is transferred to DOS memory. The other three-quarters of the image can be transferred with the function COLO640P2.

void far pascal COLO640P2 (pbuf)

char far * pbuf - pointer to an image buffer with 640x120x3 Bytes

This function transfers with each call the next 120 lines of an image that has already been captured by COLO640P1. Typically, a single call to COLO640P1 is followed by 3 calls to this function.

void far pascal COLO768P1 (pbuf)

char far * pbuf - pointer to an image buffer with 768x144x3 Bytes

This function is used to capture true-color frames with 768x576 pixels from video sources with 50 Hz-TV-standards.
Because of the limited memory for DOS applications, an image is digitized with the full size of 768x576 pixels, but only 1/4 of the image data is transferred to DOS memory. The other three-quarters of the image can be transferred with the function COLO768P2.

void far pascal COLO768P2 (pbuf)

char far * pbuf - pointer to an image buffer with 768x144x3 Bytes

Transfers the remaining three-quarters of a 768x576 image after using COLO768P1. Normally, a single call to COLO768P1 is followed by 3 calls to this function.

void far pascal DISPCOLO (pbuf, bits, xvga, yvga, ximg, yimg, xpos, ypos, xlen, ylen)

char far * pbuf	-	pointer to image buffer
int bits	-	pixel depth of VGA mode
int xvga	-	VGA x -resolution
int yvga	-	VGA y -resolution
int ximg	-	image data x -resolution
int yimg	-	image data y -resolution

int xpos	-	VGA x -position
int ypos	-	VGA y -position
int xlen	-	displayed x -resolution
int ylen	-	displayed y -resolution

This function is used for a rough display of color images for test purposes. 4- und 8-bit modes of VGA boards are supported. The 4-bit-modes use 16 color levels with resolutions of up to 640x480 pixels. On some SVGA boards this function works without problems for resolutions of 800x600 pixels. The 8-bit-mode supports the VGA resolution of 320x200 pixels. VGA boards with Tseng Labs ET4000 controller and some SVGA boards with similar memory page switching functions can work at resolutions of up to 800x600 and 1024x768. The VGA (x, y) position should be set to (0,0) for working at higher resolutions. In cases where the image is larger than the VGA resolution, the image can be cropped with the help of the parameters xlen, ylen.
This function is only a rough display for test purposes, since color images at graphic resolutions with 256 colors and less can only display high-quality images in conjunction with dithering procedures and procedures which calculate optimal color palettes for a given image.
Image data for this function must be 24 bit/pixel. For VGA resolutions with 16 colors this information is reduced to 2+1+1 bits für red, green and blue information.
For VGA resolutions with 256 colors, 3+3+2 bits for the red, green and blue channel are used.

void far pascal PALCOLO16 (void)

For VGA resolutions with 16 colors this function generates a palette as required by DISPCOLO.

void far pascal PALCOLO256 (void)

This function applies a 3+3+2 bit RGB palette as required by DISPCOLO.

void far pascal ONLINEGREY (void)

This function realizes a high-speed online display for 256 grey levels. A special FG-30 mode is used in which the image resolution is reduced 2:1. On fast computers, one achieves 25 frames/s.
The visible window of 320x200 pixels can be moved over the basic grid of 384x288 pixels with the arrow keys. The space bar interrupts this functions, and a high-resolution image can be frozen, for example.

void far pascal ONLINECOLOR (void)

This function is equivalent to ONLINEGREY except that the image display is in color. The color resolution is very low because a special mode with 4 bit luminance information, 2 bits for red color difference signal and 2 bits for blue color difference signal is applied. This mode should only be used on simple VGA boards which cannot show higher color depths. Furthermore, this mode is only useful for having a fast online display followed by a frozen image with a higher color depth.

void far pascal ONLINEPAL (r, g, b);

int r
int g
int b

This function realizes a color palette like that used by the function ONLINECOLOR.
The parameters for red, green and blue show the intensity in % to

the nominal value.

4.3. Borland C++ 3.1, 4.0, 4.5

The subdirectory DOSBLC31 contains the following files:

Name	Original	Packed	Ratio	Date	Time	Attr	CRC
DOSBLC.EXE	39824	17899	44.9%	93-07-01	02:00:00	a--w	22A4
DOSBLC31.C	21395	3397	15.9%	93-07-01	02:00:00	a--w	5342
DOSBLC31.DSK	581	286	49.2%	93-07-01	02:00:00	a--w	638B
DOSBLC31.LIB	18447	7957	43.1%	93-07-01	02:00:00	a--w	88BD
DOSBLC31.PRJ	5208	1136	21.8%	93-07-01	02:00:00	a--w	9809
DOSFG32.CFG	299	42	14.0%	93-07-01	02:00:00	a--w	3660
6 files	85754	30717	35.8%	93-07-01	02:00:00		

The functions of the library DOSBLC31.LIB are equivalent to the functions described in chapter 4.2.
For the demo program, BGI function calls are not used, so as to keep the source code simple.

4.4. Programming in Basic

4.4.1. Microsoft Quick Basic 4.5

Unfortunately Quick Basic was released with two incompatible library standards. The older standard is referred to in this document as the "English" version (DOSQB45E) and the newer standard as the "German" version (DOSQB45D). You will thus find two examples for the same Quick Basic application with the following files:
DOSQB45E:

Name	Original	Packed	Ratio	Date	Time	Attr	CRC
DOSFG32.CFG	299	40	13.4%	93-07-01	02:00:00	a--w	6530
DOSQB45.BAS	20282	3322	16.4%	93-07-01	02:00:00	a--w	5385
DOSQB45.EXE	25866	11448	44.3%	93-07-01	02:00:00	a--w	6A9D
DOSQB45E.LIB	18447	8043	43.6%	93-07-01	02:00:00	a--w	2299
DOSQB45E.QLB	23498	9768	41.6%	93-07-01	02:00:00	a--w	855E
RUN.BAT	36	36	100.0%	93-07-01	02:00:00	a--w	24BE
6 files	88428	32657	36.9%	93-07-01	02:00:00		

DOSQB45D:

Name	Original	Packed	Ratio	Date	Time	Attr	CRC
DOSFG32.CFG	299	40	13.4%	93-07-01	02:00:00	a--w	6530
DOSQB45.BAS	20282	3322	16.4%	93-07-01	02:00:00	a--w	5385
DOSQB45.EXE	66240	39283	59.3%	93-07-01	02:00:00	a--w	5AEB
RUN.BAT	36	36	100.0%	93-07-01	02:00:00	a--w	E87F
DOSQB45D.QLB	23498	9770	41.6%	93-07-01	02:00:00	a--w	9530
DOSQB45D.LIB	18447	8043	43.6%	93-07-01	02:00:00	a--w	139A
6 files	128802	60494	47.0%	93-07-01	02:00:00		

The Basic source file is the same in both versions.
A stand-alone *.EXE file is supplied only in the DOSQB45D version. The English version requires the Basic run time file BRUN45.EXE.

You need the libraries DOSQB45x.QLB to work under the Quick Basic shell. You can call Quick Basic with RUN.BAT, which sets the required options.

To work with the command-line option of this compiler you need the libraries: DOSQB45x.LIB. Both the QLB and LIB libraries contain the same functions.
For the following functions, please refer to the description of the C-functions with the same name as they are presented in section 4.1:

DECLARE SUB **LIBMAIN** ()
DECLARE SUB **SETUP** ()
DECLARE SUB **SETIMODE** (BYVAL imode AS INTEGER)
DECLARE SUB **CHECKTSENG** (SEG sm AS INTEGER)
DECLARE SUB **SWITCHTSENG** ()
DECLARE SUB **GREY320** (SEG buf AS INTEGER)
DECLARE SUB **GREY384** (SEG buf AS INTEGER)
DECLARE SUB **GREY640** (SEG buf AS INTEGER)
DECLARE SUB **GREY768** (SEG buf AS INTEGER)
DECLARE SUB **GREY320AV** (SEG buf AS INTEGER,
BYVAL av AS INTEGER)
DECLARE SUB **GREY384AV** (SEG buf AS INTEGER,
BYVAL av AS INTEGER)
DECLARE SUB **GREY640AV** (SEG buf AS INTEGER,
BYVAL av AS INTEGER)
DECLARE SUB **GREY768AV** (SEG buf AS INTEGER,
BYVAL av AS INTEGER)
DECLARE SUB **COLO320** (SEG buf AS INTEGER)
DECLARE SUB **COLO384** (SEG buf AS INTEGER)
DECLARE SUB **COLO640P1** (SEG buf AS INTEGER)
DECLARE SUB **COLO640P2** (SEG buf AS INTEGER)
DECLARE SUB **COLO768P**1 (SEG buf AS INTEGER)
DECLARE SUB COLO768P2 (SEG buf AS INTEGER)
DECLARE SUB DISPGREY (SEG buf AS INTEGER,
BYVAL bits AS INTEGER, BYVAL xvga AS BINTEGER,BY-
VAL yvga AS INTEGER, BYVAL ximg AS INTEGER,
BYVAL yimg AS INTEGER, BYVAL xpos AS INTEGER,
BYVAL ypos AS INTEGER, BYVAL xlen AS INTEGER,

BYVAL ylen AS INTEGER)

DECLARE SUB DISPCOLO (SEG buf AS INTEGER,
BYVAL bits AS INTEGER, BYVAL xvga AS INTEGER,
BYVAL yvga AS INTEGER, BYVAL ximg AS INTEGER,
BYVAL yimg AS INTEGER, BYVAL xpos AS INTEGER,
BYVAL ypos AS INTEGER, BYVAL xlen AS INTEGER,
BYVAL ylen AS INTEGER)
DECLARE SUB PALGREY16 ()
DECLARE SUB PALGREY256 ()
DECLARE SUB **PALCOLO16** ()
DECLARE SUB **PALCOLO256** ()
DECLARE SUB **ONLINEGREY** ()
DECLARE SUB **ONLINECOLOR** ()
DECLARE SUB **ONLINEPAL** (BYVAL r AS INTEGER,
BYVAL g AS INTEGER, BYVAL b AS INTEGER)

DECLARE SUB **VGATEXT** ()

This function switches the VGA board into text mode.

DECLARE SUB **VGACGA** ()

This function switches the VGA board into 320x200 graphic mode with 256 colors.

DECLARE SUB **VGAVGA** ()

This function switches the VGA board into 640x480 graphic mode with 16 colors.

4.5. Programming in Pascal

4.5.1. Borland Pascal 7.0

The subdirectory DOSPAS70 contains the following files:

Name	Original	Packed	Ratio	Date	Time	Attr	CRC
DOSFG32.CFG	299	40	13.4%	93-07-01	02:00:00	a--w	6530
DOSLIBTP.OBJ	17015	7402	43.5%	93-07-01	02:00:00	a--w	0196
DOSPAS.EXE	24480	10246	41.9%	93-07-01	02:00:00	a--w	A5D4
DOSPAS.PAS	18757	3124	16.7%	93-07-01	02:00:00	a--w	1C3F
4 files	60551	20812	34.4%	93-07-01	02:00:00		

The file FG31SVHS contains a modified version for switching FG-31 to the hosiden connector.

The following procedures are described under the same name in section 4.1.:

procedure **LIBMAIN** (x: Integer); far;
procedure **SETUP** (x: Integer); far;
procedure **SETIMODE** (imode : Integer);far;
function **CHECKTSENG** (x: Integer) : Integer; far;
procedure **SWITCHTSENG** (x: Integer); far;
procedure **GREY320** (buf: pointer); far;
procedure **GREY384** (buf: pointer); far;
procedure **GREY640** (buf: pointer); far;
procedure **GREY768** (buf: pointer); far;
procedure **GREY320AV** (buf: pointer; av: Integer); far;
procedure **GREY384AV** (buf: pointer; av: Integer); far;
procedure **GREY640AV** (buf: pointer; av: Integer); far;
procedure **GREY768AV** (buf: pointer; av: Integer); far;
procedure **COLO320** (buf: pointer); far;
procedure **COLO384** (buf: pointer); far;
procedure **COLO640P1** (buf: pointer); far;
procedure **COLO640P2** (buf: pointer); far;

procedure **COLO768P1** (buf: pointer); far;
procedure **COLO768P2** (buf: pointer); far;
procedure **DISPGREY** (buf: pointer; bits: Integer;
xvga: Integer; yvga: Integer; ximg: Integer;
yimg: Integer; xpos: Integer; ypos: Integer;
xlen: Integer; ylen: Integer); far;
procedure **DISPCOLO** (buf: pointer; bits: Integer;
xvga: Integer; yvga: Integer; ximg: Integer;
yimg: Integer; xpos: Integer; ypos: Integer;
xlen: Integer; ylen: Integer); far;
procedure **PALGREY16** (x: Integer); far;
procedure **PALGREY256** (x: Integer);far;
procedure **PALCOLO16** (x: Integer); far;
procedure **PALCOLO256** (x: Integer);far;
procedure **ONLINEGREY** (x: Integer); far;
procedure **ONLINECOLOR** (x: Integer); far;
procedure **ONLINEPAL** (r:Integer; g:Integer; b:Integer); far;

The following functions use BIOS Interrupts to switch between screen modes:

procedure **SWITCHVGA** (x: Integer);far;
Switches to standard VGA-mode: 640x480 pixels with 16 colors.

procedure **SWITCHCGA** (x: Integer);far;
Switches to standard VGA-mode: 320x200 pixels with 256 colors.

procedure **SWITCHTEXT** (x: Integer);far;
Switches the VGA board to text mode.
Unfortunatelyy Turbo Pascal uses its own procedure for text outputs.
Whenever a graphics mode is selected, Pascal functions such as WriteLn do not work correctly.

To avoid problems with different screen modes, you find the following functions, which work in all of the shown screen modes:

procedure **VGACLS** (x: Integer);far;
Clear screen 640x480

procedure **CGACLS** (x: Integer);far;

Clear screen 320x200

procedure **VGATEXTCLS** (x: Integer);far;
Clear screen in text mode.

procedure **WriteVgaLn** (x: string; attr: Integer);far;
Usable in all screen mode in the same way as the Pascal function WriteLn is used. The integer variable gives a color attribute for the string variable.

procedure **VgaInteger** (y:Integer;x:Integer;num:Integer);far;

Displays the number of n(range 0...999) at screen position (x, y).

V. Low-Level programming

The term "Low-Level programming" is used in this document to refer to parts of programs in which functions of the device driver API are called directly.
For WinMe/98/95/31 and DOS FG3xDRV.EXE is called by a 60H software interrupt.
For WinMe/98/95/31 and DOS there are no data-transfer functions in the driver. Data transfer in this case has to be done directly from an I/O port. Data comes in the form of a sequential data stream.
Under WinXP/2000/NT FG32DRV.SYSis called. It contains data-transfer functions which are described in section 1.2.

5.1. General structure of a device-driver call

The executable file FG30DRV.EXE installs a device driver to the software interrupt 60H.
The basic structure of a function call looks as follows in 80x86 assembly language:

```
mov     ax, 9709h       ;9209h for FG30
                        ;WinMe/98/95/3.x/Dos
mov     bx, funktion
mov     cx,parameter1
mov     dx,parameter2
int     60h
```

This fragment of a program can be written in various High-Level programming languages with other language-specific commands. To first get a global understanding of the meaning of these assembly language commands, you find a description below.

Each processor of an IBM PC compatible computer has, among other registers, the four basic registers AX,BX,CX and DX. Each register of this kind can operate with 16-bit-numbers. The command:

```
mov          ax,9709h     ;9209h for FG30 WinMe/98/95/3.x/Dos
```

fills the register AX with the hexadecimal value 9209. This number is the key to using all the FG30DRV functions. If other drivers use a different key and follow the Microsoft SDK standard, then multiple device drivers can share the same software interrupt 60H. Unfortunately, some network drivers do not follow these rules and can produce a lot of problems until they are correctly configured (to use a different software interrupt) or eliminated from the CONFIG.SYS file.
The register bx is loaded with a number which shows the required function of the device driver. A function of FG30DRV has up to two input parameters which can be loaded in the same manner into registers cx and dx. After a device-driver call the device-driver function can return up to 2 parameters which are contained in cx and dx after the call. The call is executed by the assembly language command

```
int          60h
```

Similar commands to get the same effect are described for various languages starting section 5.2.

To work in other operating systems in the same manner, int 60 calls are replaced by a different device-driver call.
The processor registers are now described in the form of variables.
Section 1.1.2 describes this driver call for WinXP/2000/NT along with additional data-transfer commands.

5.1.1. Overview: table of driver calls

The column “OS” shows the operating systems for which the function is implemented:

- N - Windows XP/ 2000/NT and Linux
- O - OS/2
- W - Windows Me/ 9x/ 3.xx
- D - DOS
- H - “History”, function, do not use it for new developments.

ax=9709h		Input		Output	
bx function	**OS**	**cx**	**dx**	**cx**	**dx**
00 DrvInit	NOWD	-	-	9709	vers
01 DrvPutClientX	WO	topx	lenx	-	-
02 DrvPutClientY	WO	topy	leny	-	-
03 DrvGetClientX	WO	-	-	topx	lenx
04 DrvGetClientY	WO	-	-	topy	leny
05 DrvSetAdjustment	NOWD	cl:sat ch:cont	dl:brig	-	-
06 DrvGetAdjustment	NOWD	-	-	cl:sat ch:cont	dl:brig
07 DrvSetXRAM	NOWD	index	data	-	-
08 DrvIniXRAM	NOWD	-	-	-	-
09 DrvSetBase	NOWD	basis	dwn	-	-
10 DrvSetXYoffs	NOWD	xoffs	yoffs	-	-
11 DrvSetGain	NOWD	gain	offset	-	-
12 DrvGetGain	NOWD	-	-	gain	offset
13 DrvSwitchGrabber	N	index		type 0-2	base
14 DrvDefineGrabber	N	typ,index	basis	present	-
15 DrvSetDevCapsXY	WO	xres	yres	-	-
16 DrvGetExRAM	NWD	index		-	data
17 DrvSetExRAM	NWD	index	data	-	-
18 DrvIniExRAM	NWD	-	-	-	-
19 DrvPeekExRAM	NWD	index	-	-	data
20 DrvPokeExRAM	NWD	index	data	-	-
21 Reserved	-	-	-	-	-
22 DrvGetPal	H	-	-	-	-
23 DrvPutPal	H	offs	-	-	-
24 DrvSetPalGrey	H	-	-	-	-
25 DrvSetPixBits	H	bits	-	-	-
26 DrvGetPixBits	H	-	-	bits	-
27 DrvOnlineRpt	H	-	-	status	-
28 DrvAcqColDiff	H	xres	yres	status	-
29 DrvSetColOffs	H	cxoffs	cyoffs	-	-

ax=9709h		Input		Output		
bx function	**OS**	**cx**	**dx**	**cx**	**dx**	
30 DrvGetXRAM	NOWD	index	-	-	data	
31 DrvGetColOffs	NOWD	-	-	cxoffs	cyoffs	
32 DrvAcqColDiff2	H	xres	yres	status	-	50Hz
33 DrvAcqColDiff4	H	xres/2	yres	status	-	50Hz
34 DrvSetPalCol8	H	R+G	Blue	-	-	
35 DrvAcqColDiff60	H	xres	yres	status	-	60Hz
36 DrvAcqColDiff260	H	xres	yres	status	-	60Hz
37 DrvSwitchInput	NOWD	input	-	-	-	
38 DrvGetSwSet	NOWD	input	-	swset	-	
39 DrvSetSwSet	NOWD	input	swset	-	-	
40 DrvGetInput	NOWD	-	-	input	-	
41 DrvGetBasis	NOWD	-	-	basis	-	
42 DrvGetWaits	H	-	-	waits	-	
43 DrvEingCpy	H	src	dst	-	-	
44 DrvGetCardType	NOWD			colflag	ycflag	
45 DrvAcqGreyBig	H	-	-	status	-	
46 DrvAcqGreyBig60	H	-	-	status	-	
47 DrvAcqGreySmall60	H	-	-	status	-	
48 DrvUserOutput	H	bits	-	-	-	
49 DrvSetFrameType	NOWD	frtype	-	-	-	
50 FlmSetSize	NOWD	fxsize	fysize	-	-	
51 FlmSetTopLeft	NOWD	xpos	ypos	-	-	
52 FlmSetStatus	NOWD	xstatus	ystatus	-	-	
53 FlmFirstFrame	NOWD	-	-	status	-	
54 FlmNextFrame	NOWD	-	-	status	-	
55 FlmIniNextFrame	NOWD	-	-	-	-	
56 FlmWaitNextFrame	NOWD	-	-	status	-	
57 FlmGetStatus	NOWD	mask -		status	-	
58 FlmReadBlind	NOWD	count	basis offs	-	-	
60 FlmAcq	NOWD	-	-	-	basis	
61 FlmWait	NOWD	-	-	-	basis	
62 FlmReRead	NOWD	-	-	-	basis	

ax=9709h		Input		Output	
bx function	**OS**	**cx**	**dx**	**cx**	**dx**
63 RealModeRead	D	CDwords	dx:di	-	-
64 ReadDword	NOWD	-	-	loword	hiword
80 DrvVgaDispCga	DH	-	-	-	-
81 DrvVgaIniXRAM	DH	-	-	-	-
82 DrvVgaAcq768H	DH	-	-	status	-
83 DrvVgaAcq768	DH	-	-	status	-
84 DrvVgaAcq384	DH	-	-	status	-
85 DrvVgaGetOffs	DH	-	-	offsx	offsy
86 DrvVgaSetOffs	DH	offsx	offsy	-	-
87 DrvVgaDispCgaColor	H	-	-	-	-
88 DrvVgaColImages	H	imagebuf	-	status	-
89 DrvVgaGetDither	H	-	-	dflag	-
90 DrvVgaSetDither	H	ditherflag	-	-	-
91 DrvVgaGetShift	H	-	-	sixpos	siypos
92 DrvVgaSetColPal	H	-	-	-	-
93 DrvVgaSaveBmp	H	hwndfile	segmen t	-	-
100 DrvHcDisp	H	-	-	status	-
101 DrvHcIniXRAM	H	-	-	-	-
102 DrvHcSetXRAM	H	segm	offs	-	-
103 DrvHcGetXRAM	H	segm	offs	-	-
104 DrvHcSetScreenParm	H	segm	offs	-	-
105 DrvHcGetScreenParm	H	segm	-	offs	-
106 DrvHcSetOffs	H	colxoffs	colyoffs	-	-
107 DrvHcGetOffs	H	-	-	colxoffs	colyoffs
108 DrvHcGetParm123	H	-	-	seg	offs
109 DrvHcSetImgType	H	frtype	size	-	-
110 DrvHcGetImgType	H	-	-	frtype	size
111 DrvHcDispBig	H	segment	offset	status	-
112 DrvHcDispSmall	H	segment	offset	status	-
113 DrvHcRedispBig	H	segment	offset	status	-

ax=9709h		Input		Output	
bx function	**OS**	**cx**	**dx**	**cx**	**dx**
114 DrvHcSetPosBig	H	bigshiftx	bigshifty	status	-
115 DrvSetIMode	NOWD	mode	-	-	-
116 DrvGetIMode	NOWD	-	-	mode	-

Function 0
Name of function: DrvInit

input:	ax	driver signature FG30:9209, FG31...35:9709 (MASM: 9209H Basic &H9209 Pascal: $9209 C: 0x9209)
returns:	cx	the driver signature should be returned. Any other value is a sign of an error resulting from an incorrectly installed driver. If a driver is not present, the system may hang up.
	dx	Driver version * 100

This function should be called directly after an FG-3x application is started.

Function 1
Name of function: DrvPutClientX

Input:	cx	X - position of the client window (top left corner)
	dx	X - horizontal length of client window
returns:		nothing

The driver will be informed about the X position of a client window under MS-Windows 3.x or OS/2.

Function 2
Name of function: DrvPutClientY

Input:	cx	Y - position of top left corner
	dx	Y - length of the client window
returns:		nothing

The driver will be informed about the Y position of a client window under MS-Windows or OS/2.

Function 3
Name of function: DrvGetClientX

Input:	nothing	
returns:	cx	X -position of client window
	dx	X -length of client window

This function reports the last values of the X position of a client window which have been set to the device driver.

Function 4
Name of function: DrvGetClientY

Input:	nothing	
returns:	cx	Y position of client window
	dx	Y length of client window

This function reports the last values of the Y position of a client window which have been set to the device driver.

Function 5
Name of function: DrvSetAdjustments

Input:	cl=saturation, ch=contrast, dl=brightness
returns:	nothing

8-Bit values for adjustments. The range for brightness is 0 to 255, the range for contrast is 0...127 and for saturation 0....63. Other values for saturation and contrast may reverse data. It is possible, for example, to change U and V into YUYV format.

Function 6
Name of function: DrvGetAdjustments

Input:	nothing
returns:	cl=saturation, ch=contrast, dl=brightness

The range of values returned is as shown in function 5

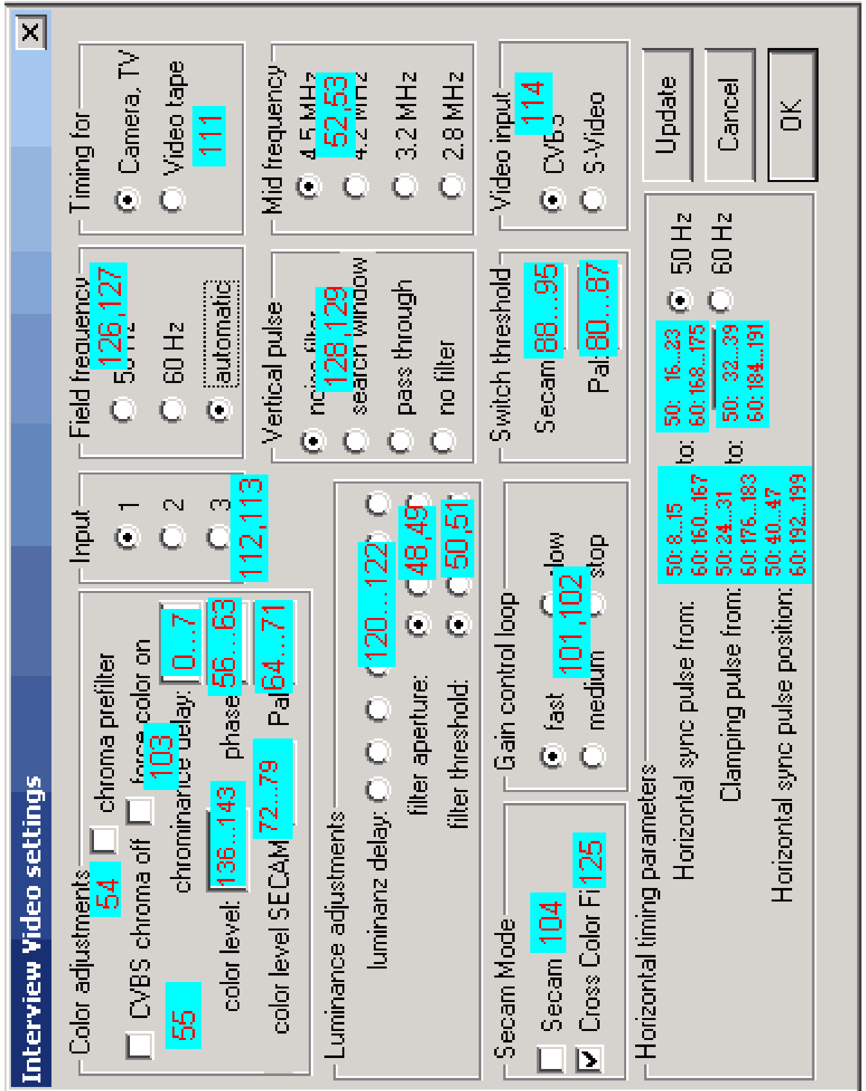
Function 7
Name of function: DrvSetXRAM

Input:	cx	XRAM address
	dl	XRAM data
returns:		nothing

This function is used to preset parameters which are used to decode color or b/w signals. A memory of 200 bits (XRAM) is used for each of the video inputs.
The address of a bit is calculated as:

position of a bit in <dl> + 8 * <cx>

The changes are made on a byte by byte basis.
The following figure of a dialog box shows the elements with their corresponding bit positions in the XRAM.



Function 8
Name of function: DrvIniXRAM

Input:		nothing
returns:		nothing

This function activates all changes made to the XRAM (e.g. with function 7) of the actual video input.

Function 9
Name of function: DrvSetBasis

Input:	cx	base address of FG-30
	dx	dwn
returns:		nothing

This function sets the base address of FG-3x. If more than one board is used in a computer, a base address switch can be used to switch between FG-3x boards. Under Windows XP/2000/NT the value of dwn is used. If dwn=1, a shorter initialization is used so as to ensure short switching times between cards. For dwn=0, a download procedure is made for FG-30-I, FG31 and FG-32. It is advisable to run this function at least one time with dwn=0 at the start of each program.

Function 10
Name of function: DrvSetXYoffs

Input:	cx	X - offset of image position
	dx	Y - offset of image position
returns:		nothing

This function allows you to adjust the position of a frame.

The offset values are related to the horizontal and vertical sync pulses. To acquire frames correctly, you must be certain that the number of lines and pixels per line used by the hardware window really exist in the video signal. If, for example, a full frame is captured, the number of y-offset must be smaller than the difference between the number of lines of the video signal and the number of lines captured. For this reason it is useful to use small offset values to test unknown video sources the first time (cx=80, dx=5). After capturing stable images, the offset value can be increased in small steps until the image position is correct.
As long as the described conditions are guaranteed, the offset values for smaller parts of the image can address a top left corner placed inside the image grid. The supplied standard software does not make use of image parts (with the exception of the 592x442 resolution in FG3xCLIP and ET4HICOL).

Function 11
Name of function: DrvSetGain

Input:	cx	gain
	dx	offset
returns:		nothing

The range of values is 0...255. The gain value is only active when gain control is switched to manual mode (use functions 30, 7, 8 to change the gain control mode).

Function 12
Name of function: DrvGetGain

Input:		nothing
returns:	cx	gain
	dx	offset

The range of values is 0...255.

Function 13
Name of function: DrvSwitchGrabber

Input:	cx	index [0..7]
returns:	cx	typ (0=FG30,1=FG31,2=FG32, 3=FG33, 4=FG34)
	dx	baseaddr

Under Windows XP/ 2000/ NT this function allows for fast switching between different boards. There is a driver internal table for 8 cards, which can be changed with function 14

card	index	type	Basisadresse
1	0	FG-32	300H
2	1	FG-30	310H
3	2	FG-31	318H
4	3	FG-32	700H
5	4	FG-33	0000H: no card present or p&p basis
6	5	FG-33	0000H: no card present or p&p basis
7	6	FG-34/35	0000H: no card present or p&p basis
8	7	FG-34/35	0000H: no card present or p&p basis

If you wish to maintain compatibility to older software, then this function must not be used. DrvSetBasis (function 9) switches to the right card if 300H,310H,318H or 700H is sent as the base address. If the plug & play (p&p) addresses of the new cards are known, this kind of switching works for cards 5...8 as well. There is an additional mechanism implemented: if function 9 switches to 300H, for example, and if there is no FG-32 on address 300H, the next existing plug & play card (card 5...8) is activated. This makes it possible to write programs for a single FG32...35, so that the first installed card can be addressed with a single function-9-call.

Function 14
Name of function: DrvDefineGrabber

input:	cl	typ (0=FG30,1=FG31,2=FG32 3=FG33, 4=FG34)
	ch	index [0...7]
	dx	baseaddr
returns:	cx	present

With this function, the list of cards can be changed. This function is only required if the predefined cards in function 13 are not sufficient. This is the case if more than 2x FG33,34,35 are used or if more than one FG-31 is used.
Before using this function you must switch to a different card index. If the selected card index is changed by this function, unexpected results may occur.

Function 15
Name of function: DrvSetDevCapsXY

Input:	cx	x resolution of actual video mode
	dx	y resolution of actual video mode
returns:		nothing

This device driver has functions for using these values to support ET4000 boards. Under MS-Windows the X resolution must be chosen carefully, because some 256-color device drivers with 800x600 organize the frame buffer with 1024 pixels per line.

Function 16
Name of function: DrvGetExRAM

Input:	cx	index (0...255) and 0ffffh
returns:	dx	data

Reads back video parameters to be modified with function 17. Offset 0ffffh is a special mode for this function. It updates all values from the current hardware status in a drivers buffer. This 8-bit-buffer can be read with offsets from 0 to 255.

Function 17
Name of function: DrvSetExRAM

Input:	cx	index (0...255)
	dx	data
returns:	nothing	

All important video parameters can be manipulated with function 7 DrvSetXRAM, function 8 DrvIniXRAM and function 30 DrvGetXRAM.
The frame grabbers FG-30-II, FG-33, FG-33-II, FG-34 and FG-35 have additional video parameters. They can be manipulated with FG3xCLIP version 4.86 or later. Using Crtl-F8, a dialog opens with multiple pages to manipulate all ExRAM parameters. This works simultaneously with the quick online display (F5).
In contrast to the XRAM functions, ExRAM is not stored for each video input. If you wish to manipulate any of these parameters, you can get the hardware status by means of function 16 with offset 0ffffh and then read the required values. The aim of function 17 is to write back these values to the buffer. The modified values must be activated with function 18.

Function 18
Name of function: DrvIniExRAM

Input:	nothing
returns:	nothing

This function activates the value set by function 17.

Function 19
Name of function: DrvPeekExRAM

Input:	cx	index
returns:	dx	data

This function reads a single ExRAM Byte directly from the hardware, bypassing the buffer of function 16.

Function 20
Name of function: DrvPokeExRAM

Input:	cx	index
	dx	data
returns:	nothing	

This function writes a single ExRAM Byte directly to the hardware, in order to achieve the effect of the change immediately.

Function 22
Name of function: DrvGetPal

Input:		nothing
returns:		nothing

Stores the current system palette. This function requires palette registers which correspond to the VGA standard.

Function 23
Name of function: DrvPutPal

Input:	cx	offset=0
returns:		nothing

Replaces the system palette which was saved with function 22.

Function 24

Name of function: DrvSetPalGrey

Input:		nothing
returns:		nothing

This function applies a VGA palette with 256 grey levels.

Function 25
Name of function: DrvSetPixBits

Input:	cx	number of bits per pixel
returns:		nothing

This function reports the color depth to the device driver. On this basis the device driver can run different routines for online displays. A pixel depth of 8 or 16 bits is supported.

Function 26
Name of function: DrvGetPixBits

Input:		nothing
returns:	cx	number of bits per pixel

This function returns the last value which was set by function 25.

Function 27
Name of function: DrvOnlineRpt

Input:		nothing
returns:	cx	0=successful grabbed

This function repeats the last online display at maximal speed. The function saves time because it avoids all initializations which are no longer necessary after a first frame has been digitized. Under Windows this function can be included in a timer-controlled loop to

update the image periodically.

Function 28 no longer present in versions V.4.1 or higher
Function name: DrvAcqColDiff

Input:	cx	x - resolution
	dx	y - resolution
returns:	cx	0= grabbed successfully

This function should not be used for new developments.
Functions 50, 51, 52, 57, 53 should be used instead:

field function/register	50cx	50dx	51cx	51dx	even 52cx	odd	52dx	interl. 57	53	2x16 pixels	bit are
592x442 YUV 50Hz	296	221	0	0		72e9h	2	yes	yes	2	YUYV
768x288 YUV 50Hz	384	288	0	0	72b9h	72c9h	2	-	yes	2	YUYV
384x288 YUV 50Hz	384	288	0	0	72b9h	72c9h	2	-	yes	1	YUYV

Up to Version 4.1:
This captures a frame in color-difference mode. After calling this function two16-bit words should be read blind to initialize the pipeline.
After this step, image data can then be read as a sequential data stream of YUV data. Every second byte represents luminance information. The rest of the information is chrominance data. The information sequence is YUYV... This sequence is known as the 4:2:2 video standard.
U is defined as a byte of green-red difference and V is defined as a byte with blue-green information.

Function 29 no longer present in versions V.4.1 or higher
Function name: DrvSetColOffset

Input:	cx	X - offset

	dx	Y - offset
returns:		nothing

For all functions that digitize color images, separate values for the image offset are set in versions 1.X. Starting with FG3xDRV Version 2.00 it is advisable to use only function 10.

Function 30
Function name: DrvGetXRAM

Input:	cx	XRAM address
	dl	XRAM data
returns:		nothing

This function is used to get parameters which are used for decoding color or b/w signals. A memory of 200 bits (XRAM) is used for each of the video inputs.
The address of a bit is calculated as:

position of a bit in <dl> + 8 * <cx>

To modify a bit of the XRAM data, the byte containing the bit should be read with this function. The next step modifies the bit(s) as required and function 7 can be used to write back the data.

Function 31
Name of function: DrvGetColOffs

Input:		nothing
returns:	cx	x - offset
	dx	y - offset

This function returns the default offset values as set by function 10.

Function 32 no longer present in versions V.4.1 or higher
Name of function: DrvAcqColDiff2

Input:	cx	x - resolution
	dx	y - resolution
returns:	cx	0=successful grabbed

This function should not be used for new developments.
Functions 50, 51, 52, 57, 53 should be used instead:

field function/register	50cx	50dx	51cx	51dx	even 52cx	odd	52dx	interl. 57	53	2x16 pixels	bit are
592x442 YUV 50Hz	296	221	0	0		72e9h	2	yes	yes	2	YUYV
768x288 YUV 50Hz	384	288	0	0	72b9h	72c9h	2	-	yes	2	YUYV
384x288 YUV 50Hz	384	288	0	0	72b9h	72c9h	2	-	yes	1	YUYV

Up to Version 4.1:
This function is similar to function 28, the only difference being that the field detection unit is turned off. The next field appearing in the video signal is thus digitized.

Function 33 no longer present in versions V.4.1 or higher
Name of function: DrvAcqColDif4

Input:	cx	x - resolution /2
	dx	y - resolution
returns:	cx	0=successful grabbed

This function should not be used for new developments.
Functions 50, 51, 52, 57, 53 should be used instead:

field function/register	50cx	50dx	51cx	51dx	even 52cx	odd	52dx	interl. 57	53	2x16 pixels	bit are
592x442 YUV 50Hz	296	221	0	0		72e9h	2	yes	yes	2	YUYV
768x288 YUV 50Hz	384	288	0	0	72b9h	72c9h	2	-	yes	2	YUYV
384x288 YUV 50Hz	384	288	0	0	72b9h	72c9h	2	-	yes	1	YUYV

Function 34
Name of function: DrvSetPalCol8

Input:	cl	red
	ch	green
	dl	blue
returns:		nothing

This function sets a color palette as required by the 8-bit-color mode. The red, green and blue values show intensity values between 0...255.

Function 35 no longer present in versions V.4.1 or higher
Name of function: DrvAcqColDif60

Input:	cx	x - resolution /2
	dx	y - resolution
returns:	cx	0=successful grabbed

This function should not be used for new developments.
Functions 50, 51, 52, 57, 53 should be used instead:

field function/register	50cx	50dx	51cx	51dx	even 52cx	odd	52dx	interl. 57	53	2x16 pixels	bit are
592x442 YUV 60Hz	296	221	-20	-4		72e9h	2	ja	ja	2	YUYV
640x240 YUV 60Hz	320	240	-20	-4	72b9h	72c9h	2	-	ja	2	YUYV
320x240 YUV 60Hz	320	240	-20	-4	72b0h	72c0h	2	-	ja	1	YUYV

Up to Version 4.1:
Similar to function 28 for 60Hz TV standards.

Function 36 no longer present in versions V.4.1 or higher
Name of function: DrvAcqColDif260

Input:	cx	x - resolution
	dx	y - resolution
returns:	cx	0=successful grabbed

This function should not be used for new developments.
Functions 50, 51, 52, 57, 53 should be used instead:

field function/register	50cx	50dx	51cx	51dx	even 52cx	odd	52dx	interl. 57	53	2x16 pixels	bit are
592x442 YUV 60Hz	296	221	-20	-4		72e9h	2	ja	ja	2	YUYV
640x240 YUV 60Hz	320	240	-20	-4	72b9h	72c9h	2	-	ja	2	YUYV
320x240 YUV 60Hz	320	240	-20	-4	72b0h	72c0h	2	-	ja	1	YUYV

Up to Version 4.1:
Similar to function 32 for 60Hz TV standards.

Function 37
Name of function: DrvSwitchInput

Input:	cx	video input to be activated
returns:		nothing

This function can select one of the video inputs. The corresponding XRAM page is also activated.

Function 38
Name of function: DrvGetSwSet

Input:	cx	input
returns:	cx	swset

This function determines for a video input the status of VCR operation (bit 1 of swset). There is no need to switch to the requested input to get the swset information.

Function 39
Name of function: DrvSetSwSet

Input:	cx	input
	dx	swset
returns:		nothing

This function can set the VCR flag for each video input.
The VCR flag is bit 1 of swset, the S-Video flag is only valid for FG30ISA and is shown with bit 0 of swset. The value 0 for a bit position turns its function off.

Function 40
Name of function: DrvGetInput

Input:		nothing
returns:	cx	input

This function returns the number of the currently selected video input (0...8).
Cx=0 is the first composite video input. Cx=2,3,5,6,8 are further composite video inputs. cx=1,4 oder 7 are S-Video inputs. If a S-Video input is used cx=n (e.g. 4), the two neighboring inputs n-1 and n+1 (3 and 5) are used – and cannot be used for composite signals at the same time. In other words: 2 composite inputs together can be used as one S-Video input.

Function 41
Name of function: DrvGetBasis

Input:		nothing
returns:	cx	base address

This function returns the actual base address used by the currently selected FG-3x board.

Function 42
Name of function: DrvGetWaits

Input:		nothing
returns:	cx	wait states (0 or 1)

This function returns the number of wait states which was defined last by device driver calls. If no value was set, the function returns the default value 1 .

Function 43 no longer present in versions V.4.1 or higher
Name of function: DrvEingCpy

Input:	cx	source (0...2)
	dx	destination (0..2)
returns:		nothing

This function copies the XRAM data block from the source input to the XRAM location of the destination input.

Function 44
Name of function: DrvGetCardType

Input:		nothing
returns:	cx	colflag

	dx	ycflag

This function shows with colflag that the board can work in color formats. The value of ycflag shows that the card can work with Y/C signals. All standard products FG30-I, FG30-II and FG31...35 return colflag=1 and ycflag=1 for yes..

Function 45 no longer present in versions V.4.1 or higher
Name of function: DrvAcqGreyBig60

Input:		nothing
returns:	cx	0=successful grabbed

This function should not be used for new developments.
Functions 50, 51, 52, 57, 53 should be used instead.

Up to Version 4.1:
This function digitizes grey-level images with 640x480 pixel resolution. After using this function, two words should be read blind to initialize the pipeline. Now the image data can be read in sequential order. Each 16-bit-word contains 2 8-bit-pixels. The first 640x240 pixels contain the odd field, the next 640x240 pixel contain the even field of the frame.

Function 47 no longer present in versions V.4.1 or higher
Name of function: DrvAcqGreySmall60

Input:		nothing
returns:	cx	0=successful grabbed

This function should not be used for new developments.
Functions 50, 51, 52, 57, 53 should be used instead:
Up to Version 4.1:
Digitizes grey-level information with a resolution of 640x240 pixels.

After reading two words to initialize the pipeline, the user has access to a 16-bit-sequential data stream. This function can be used to read a 320x240 image, if every second pixel is ignored. This is a 2:1 reduced image containing the area of a 640x480 basic grid. This function is intended for use with US-TV-standard sources.

Function 48
Name of function: DrvUserOutput

Input:	cx	bits
returns:	-	

sets the user-defined output bits. The default value of these bits is high (1).

Function 49
Name of function: DrvSetFrameType

Input:	cx	frame type
returns:		nothing

sets the frame type: 1=odd field, 2=even field 3=next field.

Function 50
Name of function: DrvFlmSetSize

Input:	cx	x - size of film image
	dx	y - size of film image
returns:		nothing

Sets the frame size for film sequences. This function only has an influence for functions 50...99.

The following table shows examples for interlaced and non-interlaced images and rectangular image parts.
The table a complete list of the formats used in FG3xCLIP (chapter 3):

field function/register	50cx	50dx	51cx	51dx	even 52cx	odd	52dx	interl. 57	53	2x16 pixels	bit are
768x576 YUV 50Hz	384	288	0	0	72e9h		2	ja	ja	2	YUYV
592x442 YUV 50Hz	296	221	0	0	72e9h		2	ja	ja	2	YUYV
768x288 YUV 50Hz	384	288	0	0	72b9h	72c9h	2	-	ja	2	YUYV
384x288 YUV 50Hz	384	288	0	0	72b9h	72c9h	2	-	ja	1	YUYV
768x576 555 50Hz	384	288	0	0	72ebh		2	ja	ja	2	2x555
384x288 555 50Hz	384	288	0	0	723fh	725fh	2	-	ja	2	2x555
768x576 565 50Hz	384	288	0	0	727bh		2	ja	ja	2	2x565
384x288 565 50Hz	384	288	0	0	72bbh	72cbh	2	-	ja	2	2x565
768x576 Grau 50Hz	384	288	0	0	72e0h		2	ja	ja	4	YYYY
768x288 Grau 50Hz	384	288	0	0	72b0h	72c0h	2	-	ja	4	YYYY
384x288 Grau 50Hz	384	288	0	0	72b2h	72c2h	2	-	ja	4	YYYY

field function/register	50cx	50dx	51cx	51dx	even 52cx	odd	52dx	interl. 57	53	2x16 pixels	bit are
640x480 YUV 60Hz	320	240	-20	-4	72e9h		2	ja	ja	2	YUYV
592x442 YUV 60Hz	296	221	-20	-4	72e9h		2	ja	ja	2	YUYV
640x240 YUV 60Hz	320	240	-20	-4	72b9h	72c9h	2	-	ja	2	YUYV
320x240 YUV 60Hz	320	240	-20	-4	723fh	725fh	2	-	ja	1	YUYV
640x480 555 50Hz	320	240	-20	-4	72ebh		2	ja	ja	2	2x555
320x240 555 50Hz	320	240	-20	-4	72bbh	72cbh	2	-	ja	2	2x555
640x480 565 50Hz	320	240	-20	-4	727bh		2	ja	ja	2	2x565
320x240 565 50Hz	320	240	-20	-4	72bbh	72cbh	2	-	ja	2	2x565
640x480 Grau 60Hz	320	240	-20	-4	72e0h		2	ja	ja	4	YYYY
640x240 Grau 60Hz	320	240	-20	-4	72b0h	72c0h	2	-	ja	4	YYYY
320x240 Grau 60Hz	320	240	-20	-4	72b2h	72c2h	2	-	ja	4	YYYY

FG-30-II, FG-33, FG-34, FG-35 have all listed formats. FG-30-I, FG-31, FG32 have no interlaced RGB and no 565 RGB format.

Function 51
Name of function: DrvFlmSetTopLeft

Input: cx x - position

dx y - position of top left corner
returns: nothing

Sets the top-left corner inside the later selected basic grid (768x576 or 640x480 or 384x288 or 320x240). This function only has an effect for functions 50...99.

For US-standards the values should be fixed to cx=-20 and dx=-4. Image adjustments should be made with function 10.

Function 52
Name of function: DrvFlmSetStatus

Input: cx x - status word
dx y - status word
returns: nothing

Sets the format for grabbing. These status words select between grey/YUV/RGB modes with 8/16 bits/pixels.
The x-status-word may have the following values.

Field:	**odd**	**even**	**next**
8 - bit - grey 1:2	72B2	72C2	72E2
8 - bit - grey 1:1	72B0	72C0	72E0
8 - bit - color 1:2	7232	7252	7272 only FG30ISA
8 - bit - color 1:1	7230	7250	7270 only FG30ISA
5+6+5-bit color 1:2	72BF	72CF	72EFnot for FG31,32
5+6+5-bit color 1:1	723F	725F	727F not for FG31,32
5+5+5-bit color 1:2	72BB	72CB	72EB
5+5+5-bit color 1:1	723B	725B	727B not for FG30ISA
YUV 1:1	72B9	72C9	72E9

Please note that resolutions grey 1:1, YUV 1:1 and 5+5+5 color 1:1 normally work in interlaced mode. The image comes in form of two fields sequentially. The sequential data transfer is possible from the I/O port of the base address of the card.

Ystatus shows the number of fields to be scanned by the grabber. It is possible to grab the next field that comes when ystatus =1. In all other cases, ystatus=2 because in the next two fields the grabber will find exactly one odd and one even field.

Function 53
Name of function: DrvFlmFirstFrame

Input:		nothing
returns:	cx	0=successful grabbed

Digitizes a first frame as defined with functions 50...52. This function can be used for 8-bit-grey-level, 8-bit-color, 16-bit-YUV and 16-bit-RGB images. Image data can be read sequentially from the following addresses:

16-bit-access (only FG-30 and possible FG-31)		
grey level:	base address	2 Pixel/ word
color 8-bit:	base address	2 Pixel/ word
color YUV:	base address	YU, YV/every other time
color RGB:	base address	(+2 for FG30ISA) RGB/ Word
32-bit-access (FG-31... FG-35)		
grey level:	base address	4 Pixel / 32-bit-word
color YUV:	base address	YUYV / 32-bit- word
color RGB:	base address	2 RGB-16-bit-Pixels / 32-bit-word

If images are grabbed in interlaced mode, both fields come in the order of their appearance in the digital data stream. To handle odd and even frames correctly, two possibilities exist:

Method 1:
Wait for each frame, start grabbing, knowing that the next field will be odd:

```
m1:     mov   ax,9709h
        mov   bx,57
        int   60h
        and   cx,2000h
        jnz   m1
```


To maintain compatibility with FG30ISA the bitmask 0010h can be used instead of 2000h.

Method 2:
Start grabbing immediately when the first field comes. The type will be detected so that it is handled as odd or even.

Function 54
Name of function: DrvFlmNextFrame

Input:		nothing
returns:	cx	0=successful grabbed
	dx	base address

Digitizes frames in the same format as function 53. This function works after a first frame has been digitized with function 53 and is faster because it avoids initializing steps needed only once.

Function 55
Name of function: DrvFlmIniNextFrame

Input:	nothing
returns:	nothing

This function performs the first step as in function 54. It does not wait for image data but returns immediately to allow software to use the time instead of waiting. Function 56 is the second part required to complete grabbing.

Function 56
Name of function: DrvFlmWaitNextFrame

Input:		nothing
returns:	cx	0=successful grabbed
	dx	base address

The grabbing process has already been started, so image data may already be present. The function waits until image data is present and prepares data for sequential reading.

Function 57
Name of function: DrvFlmGetStatus

input:	cx	-
returns:	cx	status

The returned status value contains information in some bits as described below.

Bit mask	Status information
2000H:	ODD shows the current field present in the video signal
1000H:	RDY2=0 shows that the second field is ready
0400H:	RDY=0 the first field is ready

<u>FG-32 and FG-34</u>

0100H User Input Pin 7 on Sub-D-Plug FG32 and FG-34

<u>FG-30-II and FG-33</u>

these I/O Bits have no pull-up resistors and can be used as I/O bits using the cx register. To use them as TTL-inputs, the corresponding outputs must be set to 1 (-> function 48).

0008H	User I/O of Pin 12, 15-pole-front-connector
0004H	User I/O von Pin 11, 15-pole-front-connector

Function 58 **Version 4.10 or later**
Name of function: DrvFlmBlindRead

input:	cx	count
	dx	offset to base address
returns:		nothing

This function can be used to do blind reads from I/O ports.
For FG30 16-bit-I/O-reads and for FG31...35 32-bit-I/O-reads are used.

Function 60 **Version 4.10 or later**
Name of function: DrvFlmAcq

input:		nothing
output:	cx	
	dx	base address

This function starts acquisition.

Function 61 **Version 4.10 or later**
Name of function: DrvFlmWait

input:		nothing
output:	cx	
	dx	base address

This function waits until image data is readable.

Function 62 **Version 4.10 or later**
Name of function: DrvFlmReRead

input:		nothing
output:	cx	
	dx	base address

This function starts reading. It is possible to repeat reading after this function has been used.

Function 63
Name of function: RealModeRead

input:	cx	DWORD count
	dx:di	target pointer
returns:	nothing	

This function transfers up to 16384 Dwords into a real mode 64K segment.

Function 64
Name of function: ReadDword

input:	nothing
Returns:	cx- low word, dx- high word

Function 68 **only FG-35**
Name of function: DrvSetExternalPort

Input:	cl	8-Bit-data word
	dx	offset
Returns:	nothing	

Set external data to port extender. Offset is in the range of 0...7 for one of 8 ports.

Function 69 **only FG-35**
Name of function: DrvGetExternalPort

Input:	dx	offset
Returns:	cl	8-Bit-data word

A function for reading external serial ports.
Offset is in the range of 0...7 for one of 8 ports.

Function 80
Name of function: DrvVgaDispCga

Input: nothing
Returns: nothing

Digitizes and displays grey-level images in VGA mode 13H.
The image size is reduced 2:1 and to 320x200 pixels. The visible region can be moved on the 384x288 basic grid with the help of arrow keys.

Funktion 81
Name of function: DrvVgaIniXRAM

Input: nothing
Returns: nothing

In versions later than 1.x this function is equivalent to function 7.

Function 82
Name of function: DrvVgaAcq768HalbVga

Input: nothing
Returns: cx 0= success

Digitizes grey-level images with 768x288 pixels.

Function 83
Name of function: DrvVgaAcq768

Input: nothing
Returns: cx 0= success

Digitizes grey-level images with 768x576 pixels.

Function 84
Name of function: DrvVgaAcq384

Input:	nothing	
Returns:	cx	0= success

Digitizes grey-level images with 384x288 pixels.

Function 85
Name of function: DrvVgaGetOffs

Input:	nothing	
Returns:	cx	x - Offset
	dx	y - Offset

Returns offset values placed to the driver with function 86.

Function 86
Name of function: DrvVgaSetOffs

Input:	cx	x - Offset
	dx	y - Offset
Returns:	nothing	

This function is replaced with function 10 in all versions of the software later than 2.00.

Function 87
Name of function: DrvVgaDispCgaColor

Input:	nothing
Returns:	nothing

Produces a color display using standard VGA mode 13H. Because

of the limited color capabilities and because FG-30 works only in 8-bit-color mode, this function can only be used for a preliminary display of the video source. When the image is frozen by the space bar, a function with a higher color resolution should be used to provide the requested image. During online display a 320x200 fraction of the basic grid of 384x288 can be moved by arrow keys. The color information during online display consists of 4-bit Y, 2-bit U and 2-bit V information.

Function 88
Name of function: DrvVgaColImages

Input:	cx	Segment
Returns:	cx	0= success

This function can only be used in real or virtual 386 mode of the CPU. Starting with a segment address in cx, a continuous memory area of 256 KBytes should be provided.
This function contains the following operations:

1. digitize a true color 384x288 x 24 bit RGB frame.
2. 288 x reading a line of the image to address: (segment+3000H):0000
3. 288 x color reduction of a single line to 8 bit/pixel, place results starting at: (segment+0000H):0000H
4. 288 x color reduction of a single line to 4 bit/pixel, place results starting at: (segment+2000H):0000H

Function 89
Name of function: DrvVgaGetDither

Input:		nothing
Returns:	cx	0 - no dithering, 1-dithering

Gets the dithering status for function 88.

Function 90
Name of function: DrvVgaSetDither

Input:	cx	0 - no dithering, 1 -dithering
Returns:		nothing

Sets the dithering status for function 88.

Function 91
Name of function: DrvVgaGetShift

Input:		nothing
Returns:	cx	x - position
	dx	y - position

Returns the last position of online display functions in VGA 13H mode. This position can be adjusted with the arrow keys. When replacing a frozen image with color reduction into VGA mode 13H this function shows the top-left corner of the fragment to be shown

Function 92
Name of function: DrvVgaSetColPal

Input:		nothing
Returns:		nothing

Generates and realizes a palette like the one requested directly after using function 88.

Function 93
Name of function: DrvVgaSaveBmp

Input:	cx	file handle
	dx	Segment address
Returns:		nothing

Helper function to save true color images with 384x288 pixels under DOS. This function is useful for DOS programs which do not have 324 KByte available to first capture and later to store an image into a *.BMP file.
This function assumes that an image was grabbed with function 88. After this operation the true-color image data is still in the memory of the FG-30 board. Image data access is now given line by line in the reverse order as requested by the *.BMP format.
The input value "segment address" points to a memory location with a temporary buffer of 1152 Bytes. It is assumed that a file with its file handle in cx is already opened and a valid BITMAPFILEHEADER and a BITMAPINFOHEADER is already saved. The file should be closed after using this function.

Function 100
Name of function: DrvHcDisp

Input:		nothing
returns:	cx	0= grabbed successfully

Online display of 384x288 pixels for a ET4000 graphics board working in HiColor mode with 800x600 pixels resolution.

Function 101
Name of function: DrvHcIniXRAM

Input:		nothing
returns:		nothing

Since version 1.01, this function is equivalent to Function 7.

Function 102
Name of function: DrvHcSetXRAM

Input:	cx	segment
	dx	offset
returns:		nothing

The XRAM page address which corresponds to the selected input is provided in cx:dx.

Function 103
Name of function: DrvHcGetXRAM

Input:	cx	segment
	dx	offset
returns:		nothing

The current XRAM page is copied to memory location cx:dx. The length of the copied information is 200 bits.

Function 104
Name of function: DrvHcSetScreenParm

Input:	cx	segment
	dx	offset
returns:		nothing

Updates color values for fonts and dialogs as required by the program ET4HICOL.EXE.

Function 105
Name of function: DrvHcGetScreenParm

Input:	cx	segment
	dx	offset
returns:		nothing

Reads color values for fonts and dialogs as required by the program ET4HICOL.EXE.

Function 106
Name of function: DrvHcSetOffs

Input:	cx	x - offset
	dx	y - offset
returns:		nothing

This function can be replaced by a function call to function 10.

Function 107
Name of function: DrvHcGetOffs

Input:		nothing
returns:	cx	x - offset
	dx	y - offset

returns offsets set by function 106.

Function 108
Name of function: DrvHcGetParm123

Input:		nothing
returns:	cx	segment
	dx	offset

XRAM data for all three inputs is provided for modification at real mode addresses cx:dx.

Function 109
Name of function: DrvHcSetImgType

Input:	cx	type of image
	dx	size
returns:		nothing

Defines the type of image for ET4HICOL.EXE type=0: reserved, type=1: odd frame, type=2: even frame, type=3: next field whatever appears first. Defines the image size as used in ET4HICOL size=0: 384x288 size=1: 592x442 interlaced mode 0 and size=2: 592x442 interlaced mode 1.

Function 110
Name of function: DrvHcGetImgType

Input:		nothing
returns:	cx	type of image
	dx	size

Returns the last settings made with function 109.

Function 111
Name of function: DrvHcDispBig

Input:	cx	segment
	dx	offset
returns:	cx	0=successful grabbed

Digitize and display an 592x442 image in ET4000 HiColor Mode with a resolution of 800x600.
24-bit-RGB data is captured, where the last 3 bits of each color are placed starting at address cx:dx. These values have to be stored to save true-color images which can then be combined with information which appears on screen.

Function 112
Name of function: DrvHcDispSmall

Input:	cx	segment
	dx	offset
returns:	cx	0= grabbed successfully

Digitizes and displays in ET4000 HiColor 800x600 mode an 384x288 image. At the same time, the 24-bit image data is stored at cx:dx

Function 113
Name of function: DrvHcRedispBig

Input:	cx	segment
	dx	offset
returns:		nothing

Repaints an 592x442 true-color image from data which is still available in the FG-30 frame buffer. As in function 111, the last 3 bits of each color channel are placed to location cx:dx.

Function 114
Name of function: DrvHcSetPosBig

Input:	cx	x offset
	dx	y offset
returns:		nothing

The relative screen position of a true-color image with the size 592x442 as produced by function 111 is defined by this function.

Function 115

Name of function: DrvHcInterlacedMode

Input: cx mode
returns: nothing

This function can change the interlaced mode value. Depending upon which grabber is in use, the default value is either 2 or 3. A range from 1...5 is allowed.

Function 116
Name of function: DrvGetIMode

Input: nothing
Returns: cx Mode

Returns the interlaced mode value as set by function 115.

5.2. Using driver calls

5.2.1. Microsoft Visual C++ 1.0... 1.52
5.2.2. Microsoft C/C++ 7.0

One way to realize a driver request is to use an inline assembler. Inline assemblers are provided with many C-compilers.
Under MS-Windows 3.0, 3.1 or 3.11 the DPMI interface is used. To open this interface in Win3.x the lines printed in red type below are required. For Win95 and later, the red lines are not required.

```
void DrvInit()
{
   _asm{

      mov   ax, 0200h         ;get real mode interrupt vector
      mov   bl, 60h           ;requested interrupt
      int   31h               ;DPMI call
      or    cx, dx            ;error ?
      jz    short nodpmi
      mov   ax, 9709h         ;signature FG3x, use 9209 for FG-30
      mov   bx,0              ; initialize FG30DRV
      int   60h
      cmp   bx, 9709h         ;bx = 9709 means success
      jz    dpmiok
      mov   installed, -1     ;
dpmiok:
}
}
```

Installed is a C-variable with type int. Before the call is made it can be initialized, e.g. with 0. The value of -1 shows the following procedures that the driver was not successful initialized.

For WinMe/98/95 and DOS it suffices to initialize the card this way:

```
_asm    {
        mov   bx,0
        mov   ax,9709h
        int   60h
        }
```

It is absolutely necessary that FG3xDRV.EXE be called from autoexec.bat at system startup. This call can be performed without the frame grabber being installed.

Image data comes in the form of sequential data streams. UsingFG-31...35 they come from the first 32-Bit-I/O-Port and using FG-30 they come from the first 16-Bit-I/O-Port address. To read 16-bit-I/O ports, most compilers have the inpw- function. If a compiler has no 32-bit I/O read command and no 32-bit-inline assembler a driver function can be used instead:

```
void ReadBuffer (pbuffer, maxbuffer, basis)
DWORD  far * pbuffer;
int maxbuffer, basis;
{
for (i=0;i<maxbuffer;i++)   *pbuffer++ = inp_dword (basis);
}
DWORD inp_dword (basis)
int basis;
{
int hi;
int lo;
_asm    {
        mov   ax,9709h
        mov   bx,64
        int   60h
```



```
        mov   hi,dx
        mov   lo,cx
        }
return ((DWORD)hi<<16+(DWORD)lo);
}
```

The following C 6.0 example can be used as well.

5.2.3. Microsoft C PDS/6.0
5.2.4. Microsoft Quick C 2.5
5.2.5. Microsoft Quick C for Windows

```
#include <dos.h>

union REGS inreg,outreg;

int   DrvInit ()
{
inreg.x.ax = 0x9709;          /* API */
inreg.x.bx = 0;               /* function 0 */
int86 (0x60,  &inreg,  & outreg);
if (outreg.x.cx+outreg.x.bx != 0)
 return 1;
 else
 return 0;
}
```

Read sequential 32-bit data:

```
void ReadBuffer (pbuffer, maxbuffer, basis)
int  far * pbuffer;
int maxbuffer, basis;
{
for (i=0;i<maxbuffer;i++)
```

```
        {
        inreg.x.ax=0x9709;
        inreg.x.bx=64;
        int86 (0x60, &inreg, &outreg;
        *pbuffer++ = outreg.x.cx;
        *pbuffer++ = outreg.x.dx;
        }
}
```

5.2.6. Borland C++ 3.1, 4.0, 4.5

This compiler can use a built-in inline assembler or it can use commands similar to those described in the previous chapter. The Inline-Assembler has some differences if compared to that of Microsoft C. Comments must use the C-syntax, and labels can only be placed outside of assembly language segments.

The modified example from 7.3.2.1. is shown below:

```
void DrvInit()
{
installed=0;
   asm      {

            mov   ax, 0200h    /*get real mode interrupt vector*/
            mov   bl, 60h      /*interrupt of choice*/
            int   31h          /*DPMI call */
            or    cx, dx       /*error ? */
            jz    short nodpmi
            mov   ax, 9709h    /*API indicator*/
            mov   bx,0         /*initialize FG3xDRV */
            int   60h
            cmp   bx, 9709h    /*indicator = bx: success */
```



```
            jnz   short nodpmi
            }
  installed=-1;
  nodpmi:                        /*label in C-segment */
}
```

5.3. Microsoft Quick Basic

To read sequential data, Quick Basic requires a small library. Quick Basic has the command INP(port%). But this command is of limited use since it can read only 8-bit-data from port addresses in the range of 0...255.

The following steps implement for all Quick Basic versions from 3.0 to 4.5 a new function called INPW:

Somewhere at the beginning of the program the following declaration should be included:

```
DECLARE SUB  INPW  (BYVAL basis AS INTEGER,
                         SEG iword AS INTEGER)
```

A buffer with 2048 words (16-bit) can be read as shown here:

```
DIM Buffer% (2048)
DIM INARY%(7), OUTARY% (7);
AXREG%=0
BXREG%=1
CXREG%=2
DXREG%=3

INARY%(AXREG%)=&H9709
INARY%(BXREG%)=64

FOR  i%=0 TO 1023
  CALL INT86 (&H60, VARPTR (INARY%(0)), VARPTR(OUTARY%(0)))
  Buffer% (i%*2)=OUTARY%(CXREG%)
  Buffer% (i%*2+1)=OUTARY%(DXREG%)
NEXT i%
```

The file on the supplied disk named QBAS45.EXE contains the

following files in compressed form:

INPW.ASM-	source code for Quick Library
INPW.OBJ-	Object code for Quick Library, (for the case that you don't have an Assembler)
INPW.QLB-	Quick Library

As described earlier - Quick Basic has inconsistent library formats. To be sure that you use a compatible quick library you may follow the next steps to produce a new library based on your QB45 version:

1. Copy INPW.OBJ into the directory where the Quick Basic linker LINK.EXE is located,

2. Relink the library with:

   ```
   link      /QU  inpw, , ,  lib\bqlb45.lib
   ```

3. Start Quick Basic now using:

   ```
   qb   /l inpw.qlb
   ```

For Quick Basic (version 3.0 or higher), the initializing procedure would look like this:

```
DIM INREG%(7), OUTREG(7)

AX% = 0              'Index definition for the necessary
BX% = 1              'Registers
CX% = 2
DX% = 3

INREG%(AX%) = &H9709   'Identification
```

```
INREG%(BX%) = 0                'Function 0
CALL INT86 (&H60, VARPTR ( INREG%(0)), VARPTR
 ( OUTREG%(0))))
```

Version 4.5:

Load Quick Basic together with the provided library QB.LIB. In this case you can use the function CALL INTERRUPT.
The following example uses the function CALL INT86OLD:

```
$INCLUDE: 'QB.BI'

DIM INREG%(7), OUTREG%(7)
CONST  AX=0, BX=1, CX=2, DX=3

INREG%(AX)= &H9709             'Kennung
INREG%(BX)= 0
CALL INT86OLD (&H60, INREG%(), OUTREG%())
```

To initialize the DPMI interface:

```
TYPE  RegType
 ax   AS INTEGER
 bx   AS INTEGER
 cx   AS INTEGER
 dx   AS INTEGER
 bp   AS INTEGER
 si   AS INTEGER
 diI   AS INTEGER
 flags INTEGER
 ds AS INTEGER
 es AS INTEGER
END TYPE
```



```
RegType  rgi, rgo

installed% = 0
rgi.ax = &H200                'Get real mode interrupt vector
rgi.bx = &H60                 'interrupt-handle for requested int
CALL   INTERRUPT   (&H31, rgi, rgo)              'int 31h
IF   rgo.cx + rgo.dx = 0  THEN  GOTO nodpmi      'Error?
rgi.ax = &H9709                                  'Check API
rgi.bx = 0                                       'fkt 0:  init
CALL   INTERRUPT (&H60, rgi, rgo)
IF  rgo.bx <> &H9709   THEN  GOTO  nodpmi installed% = 1
.... continue with the successfully activated driver
nodpmi:
.... handle error
```

5.4. Microsoft Visual Basic

It is best not to use this program for programming low-level functions. It is recommended that you use another language to write all necessary functions in the form of a DLL. Such DLL's are then easy to handle under Visual Basic. The provided example WINMSVB shows how to operate with functions which are located in a DLL.

5.5. Microsoft Macro-Assembler 6.0
5.6. Microsoft Macro-Assembler 5.1
5.7. Borland Turboassembler

If cmacros.inc is included in your source file with this line:

```
INCLUDE  CMACROS.INC
```

you can generate procedures for all memory models for all high-level languages.

An example of a function to activate the DPMI interface and to initialize the driver would look like this:

```
;.........................................................................
; drvini
; Aufruf von C/C++:
;       void  drvinit  (int far *  lpinstalled)
;.........................................................................
cProc         drvinit, < PUBLIC,FAR,PASCAL>,<ds>
       parmD         lpinstalled

cBegin
      lds     di,lpinstalled
      pusha
      mov     ds:[di], word ptr 0
      push    ds
      push    di
      mov     ax, 0200h          ;get real mode interrupt vector
      mov     bl, 60h            ;requested interrupt handler
      int     31h                ;DPMI call
      or       cx, dx            ;error ?
      jz      nodpmi
      mov     ax, 9709h          ;API indicator
      mov     bx,0               ; initialize FG3xDRV
      int     60h
      pop     di
      pop     ds
      cmp     cx, 9709h          API indicator in cx: success
      jnz     nodpmi
      mov     ds:[di], word ptr 1
nodpmi:
      popa
cEnd
```


5.8. Turbo Pascal for DOS
5.9. Turbo Pascal for Windows

This language has a built-in Inline Assembler. To change the comments into the syntax used under Pascal, you can use the example shown in the previous section.

Sequential data sets for FG-30 can be read as follows:

```
procedure  ReadBuffer8Bit (xres,yres,basis : integer;);
var    buffer: array[1..xres,1..yres] of integer;
 x,y   : integer;


begin
for   y:=1   to   yres+1  do
 begin
 for  x:=1 to xres+1  do
       begin
       puffer  [x,y] := portw [basis];
       end
 end
.
.
end;
```

Please note that the variable xres must contain only half of the x-resolution. Each port access reads 16 - bit - words, each containing 2 pixels.

VI.
Low-Level programming examples

6.1. Low-level programming in C

msvc-low-level-555	src-msvc-low-level-555
msvc-low-level-555-interlaced	src-msvc-low-level-555-interlaced
msvc-low-level-555-interlaced-us	src-msvc-low-level-grey
msvc-low-level-555-us	src-msvc-low-level-grey-interlaced
msvc-low-level-grey	src-msvc-low-level-us-555
msvc-low-level-grey-interlaced	src-msvc-low-level-us-555-interlaced
msvc-low-level-grey-interlaced-us	src-msvc-low-level-us-grey
msvc-low-level-grey-us	src-msvc-low-level-us-grey-interlaced

Often only one defined image format (with a few adjustments) is required. The use of direct driver calls may have some advantages, because no other components are required.
In the start menu there is a list of examples such as that shown above, where the name indicates which compiler is used and the image format. The name Msvc-low-level indicates that the Microsoft Visual C compiler (Version 2.0-6.0 and .NET) is used. 555 means that color images with 32K colors, 16 bit/Pixel are used. Grey means that the images are 8-Bit grey-scale. Interlaced means that a full frame with a resolution of 768x576 Pixels is used, Interlaced-us is the same full frame for US-standard- (60Hz-) video sources with 640x480 pixels. When the term " interlaced" is not part of the name, resolutions of 384x288 pixels are used and in US-standard 320x240 pixels.
The camera icon is used for the executable files, other symbols point to the projects files. If a compiler is installed, this project file normally opens the compiler with the corresponding source code example. You should test your compiler environment by compiling the unchanged example once and then comparing the result with the supplied executable file.

Starting with Version 4.81 the FG32...35 examples have an additional version for Borland C++ Builder 6. There are slight differences between the compilers, particularly under WinMe/9x. Differences in how the Inline-Assembler handles the source code lead to instability in the Borland compiler. The Borland examples have tested code with additional assembler statements so as to achieve the same stability as the Microsoft examples have. There are additional

examples starting with version 4.81 as shown below. They are also included for Microsoft Visual C++. The "...low-level-parameters..." examples show how to handle adjustments for the frame grabber. It is possible to load, save and edit parameter sets. The " ...low-level-dual-grabber-555..." example shows how the Windows XP/2000/NT driver can handle two frame grabber images simultaneously. The example can handle sources in US-standard or 50Hz standards with 640x480 pixels and is easy adaptable for other resolutions.

6.1.1. Low-Level programming in C for WinXP/2000/NT

The WM_CREATE procedure allocates memory for a Device Independent Bitmap (DIB). Since the image size is known, the required amount of memory can be calculated and the DIB parameters can be initialized.
The Device Driver FG32DRV.SYS is opened and the first initializing driver functions are called:

```
FG30DRV (0,&cx,&dx);          //init device driver
 cx=baseaddr;
 dx=0;                        //enable download
 FG30DRV (9,&cx,&dx);         //set baseaddr
 FG30DRV (8,&cx,&dx);         //init grabber
```

Function 0 must be called each time the program is started. Function 9 sets a base address. The boards FG30...32 operate in a fixed I/O address space, so function 9 can be used to switch between cards. To configure FG-30-I, FG-31 and FG32 downloads are required. They are required only once after the system start, and they take about a second. If function 9 is called with dx not equal zero, a fast switch is called without configuring the card again.
The frame grabber FG-30-II does not require a download. FG-33, FG34 and FG35 are plug&play boards, which are configured by the system. They do not require downloads. If it is known that the address space 300H...30FH is not used by another device, you can use the following statements instead of function 9:

```
 cx=300H;
 dx=0;                        //enable download
 FG30DRV (9,&cx,&dx);         //set baseaddr
 FG30DRV (41,&cx,&dx);        //get baseaddr
basis=cx;
```

In this case the driver tries a download on address 300H first. If a frame grabber FG-32 is present, it will be configured and function 41 will report the base address 300H. If no FG-32 is present, the driver checks for FG-33, FG34 or FG35. If at least one of these frame grabbers is installed, the driver activates the first device found and so function 41 will report the correct base address for FG32,33,34 or 35, respectively. This makes it possible to write a program that will work without modification, if at least one FG32...35 is installed. Low-level examples for FG32...35 have the same source code installed using this feature.

Function 8 initializes video parameters. This is required at least one time after a system start. It is possible to change XRAM-values before function 8 is executed. If these values are later changed, use this function to activate them again.

The acquisition process is controlled by functions 50, 51, 52 and 53. “Acquisition process” means here the process of digitizing the video signal and taking images and image parts into the frame grabbers built-in dual port memory. To get image data, you must follow a data transfer procedure.

Functions 50, 51 and 52 select a rectangular image part and the data format. To get the full resolution you must work in interlaced Mode (1:1).
1:2 down-scaled images are based on image fields, and it makes sense to handle this case with a separate example. For each TV standard this results in two basic grids: for 50Hz this is 768x576 and 384x288 and for 60 Hz it is 640x480 and 320x240.
Inside this basic grid, a rectangle can be defined to get only data found inside of it. Only this data is taken into the built-in memory and therefore only this data is transferred, which means that the data transfer time is reduced. With this functionality a higher quality is possible than that achieved using a hardware scaler. The following example tries to explain why the scaling functions of the FG-30-II, FG33...35 chipsets are not used. For example, a target image size of 280x200 pixels is to be acquired. To solve this, a basic grid of 384x288 (50Hz) or 320x240 (60Hz) is used and a rectangular image part of 280x200 is adjusted. Instead of scaling down from 384x288 (or 320x240) to 280x200, the camera can be brought closer to the object, so that only the relevant objects are contained in the cropped window.

If Interlaced mode is used, the examples use function 57 to start grabbing in an odd field, so as to get the fields in the expected order.

The image data transfer functions are described in section 1.1 of this chapter.

Image display is realized with a Windows API function SetDIBBitsToDevice. Using the functions of DirectX or DrawDibDraw a faster display is possible, which is also scalable.
Two more examples, msvc-low-level-parameters and msvc-low-level-parameters-us, show a simple live-video display realized by a timer function. It allows you to load, modify and save video parameters simultaneously.

6.1.2. Low-Level programming in C for WinMe/98/95

All examples described in the previous section are available under WinMe/98/95 as well. On these platforms the device driver must not be opened.
All int- 60h- calls are explained in section 5.1.
The image data transfer is made directly with the help of block transfer commands, such as insd (32-bit) and insw (16-bit-FG30). Such commands allow the transfer of a predefined number of DWORD (insw:WORDS).
For 1:2 image data formats, a single command will transfer the image. For interlaced formats the transfer is more complex. This will be explained using the example “low-level-interlaced-555":
Data transfer from dual-port memory follows the same order as data is presented in the video signal. The odd field comes first and is transferred with:

```
        mov     edi,pimg
        mov     ecx,288
odd00:  push    ecx
        mov     ecx,768/2
        rep     insd        ;fill buffer with image data
        add     edi,768*2
        pop     ecx
        loop    odd00
```

The frame buffer is skipped for every second line. Rep insd transfers in each cycle 384 Dwords, which are 768 16-bit-pixels. The add command skips 384 Dwords, which is equal the space of one line. This cycle is repeated 288 times. Between video fields the hardware digitizes two more lines. These lines must be read blind so as to reach the beginning of the next field:

```
        mov     ax,9709h
        mov     bx,116
        int 60h             ;drvgetimode
        and     cx,cx
```

```
                jz      rb01
readblind:      mov     bx,384          ;384 dwords=1
zeilerb00:              in      eax,dx
                dec     bx
                jnz     rb00
                dec     cx
                jnz     readblind
rb01:
```

Function 116 returns the value 2 by default. There are several video signals for which another value (settable by function 115) results in the correct placement of the even field. It makes the most sense to use values in the range of 0...4.

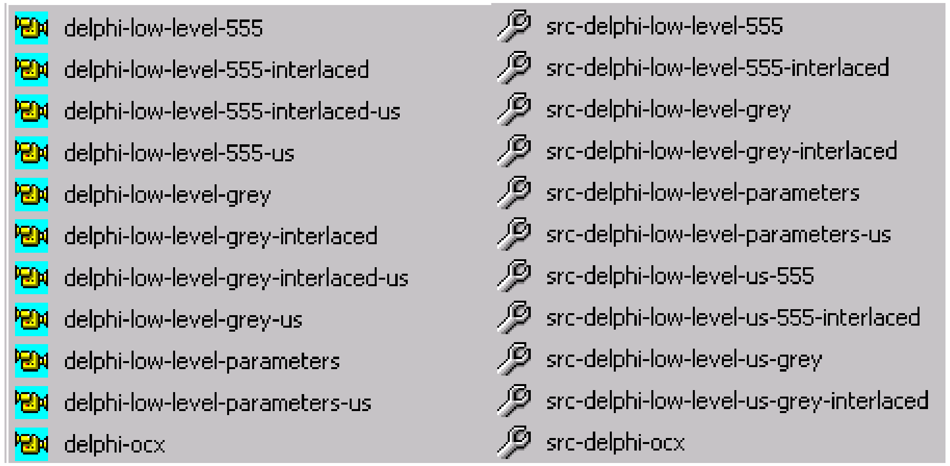
The second field is now transferred:

```
                mov     edi,pimg
                mov     ecx,288
even00:         push    ecx
                mov     ecx,768/2
                add     edi,768*2
                rep     insd        ;fill buffer with image data
                pop     ecx
                loop    even00
```

The order of insd and add edi,384 is reversed, so now the lines skipped in the first loop are filled.
As in the previous examples for Windows XP/2000/NT, a simple display of the image is implemented using SetDibBITsToDevice. With small changes, the example is useable for Windows 3.x. In this case, the order of lines must be placed into the DIB in reverse order and the negative sign of pbi->bmiHeader.biHeight must be removed. Because 16-Bit-data DIBs are not implemented in Win3.x, the DIB format must be changed to 24 bits/pixel.

6.2. Low-Level programming in Pascal

The examples shown in section 6.1.1. are installed for Borland Delphi as well. The examples are compiled with Delphi 6 but should work under Delphi 5 too. For older Delphi versions the projects must be redefined and some displaying



functions have to be changed. The frame grabber functions do not require any changes.