GRX2 User's Manual

GRX2 User's Manual

Credits

`GRX 2.4.7 User's Manual`

`A graphics library for DOS, Linux, X11 and Win32`

Based on the original doc written by: Csaba Biegl on August 10, 1992 Updated by: Mariano Alvarez Fernandez on August 17, 2000

Last update: March 4, 2003

`Abstract`

GRX is a 2D graphics library originaly written by Csaba Biegl for DJ Delorie's DOS port of the GCC compiler. Now it support a big range of platforms, the main four are: DOS (DJGPPv2), Linux console, X11 and Win32 (Mingw). On DOS it supports VGA, EGA and VESA compliant cards. On Linux console it uses svgalib or the framebuffer. On X11 it must work on any X11R5 (or later). From the 2.4 version, GRX comes with a Win32 driver, but you must considere it in alpha stage. The framebuffer Linux console driver was new in 2.4.2 and it is in alpha stage too. From 2.4.7 there is a support for x86_64 bits Linux machines and on MingW and X11 support for an SDL driver.

Hello world

The next program draws a double frame around the screen and writes "Hello, GRX world" centered. Then it waits after a key is pressed.

```
#include <string.h>
#include <grx20.h>
#include <grxkeys.h>

int main()
{
  char *message = "Hello, GRX world";
  int x, y;
  GrTextOption grt;

  GrSetMode( GR_default_graphics );

  grt.txo_font = &GrDefaultFont;
  grt.txo_fgcolor.v = GrWhite();
  grt.txo_bgcolor.v = GrBlack();
  grt.txo_direct = GR_TEXT_RIGHT;
  grt.txo_xalign = GR_ALIGN_CENTER;
  grt.txo_yalign = GR_ALIGN_CENTER;
  grt.txo_chrtype = GR_BYTE_TEXT;
  GrBox( 0,0,GrMaxX(),GrMaxY(),GrWhite() );
  GrBox( 4,4,GrMaxX()-4,GrMaxY()-4,GrWhite() );

  x = GrMaxX()/2;
  y = GrMaxY()/2;
  GrDrawString( message,strlen( message ),x,y,&grt );

  GrKeyRead();

  return 0;
}
```

How to compile the hello world (assuming the GRX library was previously installed)

```
DJGPP: gcc -o hellogrx.exe hellogrx.c -lgrx20
Mingw: gcc -o hellogrx.exe hellogrx.c -lgrx20 -mwindows
X11  : gcc -o hellogrx hellogrx.c -D__XWIN__ -I/usr/X11R6/include
       -lgrx20X -L/usr/X11R6/lib -lX11
Linux: gcc -o hellogrx hellogrx.c -lgrx20 -lvga

For the SDL driver:
Mingw: gcc -o hellogrx.exe hellogrx.c -lgrx20S -lSDL
X11  : gcc -o hellogrx hellogrx.c -D__XWIN__ -I/usr/X11R6/include
       -lgrx20S -lSDL -lpthread -L/usr/X11R6/lib -lX11

For x86_64 systems add -m32 or -m64 for 32/64 bits executables
and replace /lib by /lib64 as needed
```

Data types and function declarations

All public data structures and graphics primitives meant for usage by the application program are declared/prototyped in the header files (in the 'include' sub-directory):

```
 * grdriver.h   graphics driver format specifications
 * grfontdv.h   format of a font when loaded into memory
 * grx20.h      drawing-related structures and functions
 * grxkeys.h    platform independent key definitions
```

User programs normally only include see [grx20.h], page 154 and see [grxkeys.h], page 191

Setting the graphics driver

The graphics driver is normally set by the final user by the environment variable GRX20DRV, but a program can set it using:

```
int GrSetDriver(char *drvspec);
```

The drvspec string has the same format as the environment variable:

```
<driver> gw <width> gh <height> nc <colors>
```

Available drivers are for:

```
* DOS => herc, stdvga, stdega, et4000, cl5426, mach64, ati28800, s3, VESA, memory
* Linux => svgalib, linuxfb, memory
* X11 => xwin, memory
* Win32 => win32, memory
* SDL (Win32 and X11) => sdl::fs, sdl::ww, memory
```

The optionals gw, gh and nc parameters set the desired default graphics mode. Normal values for 'nc' are 2, 16, 256, 64K and 16M. The current driver name can be obtained from:

```
GrCurrentVideoDriver()->name
```

Setting video modes

Before a program can do any graphics drawing it has to configure the graphics driver for the desired graphics mode. It is done with the GrSetMode function as follows:

```
int GrSetMode(int which,...);
```

On succes it returns non-zero (TRUE). The which parameter can be one of the following constants, declared in grx20.h:

```
typedef enum _GR_graphicsModes {
  GR_80_25_text,
  GR_default_text,
  GR_width_height_text,
  GR_biggest_text,
  GR_320_200_graphics,
  GR_default_graphics,
  GR_width_height_graphics,
  GR_biggest_noninterlaced_graphics,
  GR_biggest_graphics,
  GR_width_height_color_graphics,
  GR_width_height_color_text,
  GR_custom_graphics,
  GR_width_height_bpp_graphics,
  GR_width_height_bpp_text,
  GR_custom_bpp_graphics,
  GR_NC_80_25_text,
  GR_NC_default_text,
  GR_NC_width_height_text,
  GR_NC_biggest_text,
  GR_NC_320_200_graphics,
  GR_NC_default_graphics,
  GR_NC_width_height_graphics,
  GR_NC_biggest_noninterlaced_graphics,
  GR_NC_biggest_graphics,
  GR_NC_width_height_color_graphics,
  GR_NC_width_height_color_text,
  GR_NC_custom_graphics,
  GR_NC_width_height_bpp_graphics,
  GR_NC_width_height_bpp_text,
  GR_NC_custom_bpp_graphics,
} GrGraphicsMode;
```

The GR_width_height_text and GR_width_height_graphics modes require the two size arguments: int width and int height.

The GR_width_height_color_graphics and GR_width_height_color_text modes require three arguments: int width, int height and GrColor colors.

The GR_width_height_bpp_graphics and GR_width_height_bpp_text modes require three arguments: int width, int height and int bpp (bits per plane instead number of colors).

The GR_custom_graphics and GR_custom_bpp_graphics modes require five arguments: int width, int height, GrColor colors or int bpp, int vx and int vy. Using this modes you can set a virtual screen of vx by vy size.

A call with any other mode does not require any arguments.

The GR_NC_... modes are equivalent to the GR_.. ones, but they don't clear the video memory.

Graphics drivers can provide info of the supported graphics modes, use the next code skeleton to colect the data:

```
{
  GrFrameMode fm;
  const GrVideoMode *mp;
  for(fm =GR_firstGraphicsFrameMode; fm <= GR_lastGraphicsFrameMode; fm++) {
    mp = GrFirstVideoMode(fm);
    while( mp != NULL ) {
      ..
      .. use the mp info
      ..
      mp = GrNextVideoMode(mp))
    }
  }
}
```

Don't worry if you don't understand it, normal user programs don't need to know about FrameModes. The GrVideoMode structure has the following fields:

```
typedef struct _GR_videoMode GrVideoMode;

struct _GR_videoMode {
  char    present;                    /* is it really available? */
  char    bpp;                        /* log2 of # of colors */
  short   width,height;               /* video mode geometry */
  short   mode;                       /* BIOS mode number (if any) */
  int     lineoffset;                 /* scan line length */
  int     privdata;                   /* driver can use it for anything */
  struct _GR_videoModeExt *extinfo;   /* extra info (maybe shared) */
};
```

The width, height and bpp members are the useful information if you are interested in set modes other than the GR_default_graphics.

A user-defined function can be invoked every time the video mode is changed (i.e. GrSetMode is called). This function should not take any parameters and don't return any value. It can be installed (for all subsequent GrSetMode calls) with the:

```
void GrSetModeHook(void (*hookfunc)(void));
```

function. The current graphics mode (one of the valid mode argument values for GrSetMode) can be obtained with the:

```
GrGraphicsMode GrCurrentMode(void);
```

function, while the type of the installed graphics adapter can be determined with the:

```
GrVideoAdapter GrAdapterType(void);
```

function. GrAdapterType returns the type of the adapter as one of the following symbolic constants (defined in grx20.h):

```
typedef enum _GR_videoAdapters {
  GR_UNKNOWN = (-1),     /* not known (before driver set) */
  GR_VGA,                /* VGA adapter */
  GR_EGA,                /* EGA adapter */
  GR_HERC,               /* Hercules mono adapter */
  GR_8514A,              /* 8514A or compatible */
  GR_S3,                 /* S3 graphics accelerator */
  GR_XWIN,               /* X11 driver */
  GR_WIN32,              /* WIN32 driver */
  GR_LNXFB,              /* Linux framebuffer */
  GR_SDL,                /* SDL driver */
  GR_MEM                 /* memory only driver */
} GrVideoAdapter;
```

Note that the VESA driver return GR_VGA here.

Graphics contexts

The library supports a set of drawing regions called contexts (the GrContext structure). These can be in video memory or in system memory. Contexts in system memory always have the same memory organization as the video memory. When GrSetMode is called, a default context is created which maps to the whole graphics screen. Contexts are described by the GrContext data structure:

```
typedef struct _GR_context GrContext;

struct _GR_context {
  struct _GR_frame    gc_frame;       /* frame buffer info */
  struct _GR_context *gc_root;        /* context which owns frame */
  int    gc_xmax;                     /* max X coord (width  - 1) */
  int    gc_ymax;                     /* max Y coord (height - 1) */
  int    gc_xoffset;                  /* X offset from root's base */
  int    gc_yoffset;                  /* Y offset from root's base */
  int    gc_xcliplo;                  /* low X clipping limit */
  int    gc_ycliplo;                    /* low Y clipping limit */
  int    gc_xcliphi;                    /* high X clipping limit */
  int    gc_ycliphi;                    /* high Y clipping limit */
  int    gc_usrxbase;                   /* user window min X coordinate */
  int    gc_usrybase;                   /* user window min Y coordinate */
  int    gc_usrwidth;                   /* user window width  */
  int    gc_usrheight;                  /* user window height */
# define gc_baseaddr                    gc_frame.gf_baseaddr
# define gc_selector                    gc_frame.gf_selector
# define gc_onscreen                    gc_frame.gf_onscreen
# define gc_memflags                    gc_frame.gf_memflags
# define gc_lineoffset                  gc_frame.gf_lineoffset
# define gc_driver                      gc_frame.gf_driver
};
```

The following four functions return information about the layout of and memory occupied by a graphics context of size width by height in the current graphics mode (as set up by GrSetMode):

```
int GrLineOffset(int width);
int GrNumPlanes(void);
long GrPlaneSize(int w,int h);
long GrContextSize(int w,int h);
```

GrLineOffset always returns the offset between successive pixel rows of the context in bytes. GrNumPlanes returns the number of bitmap planes in the current graphics mode. GrContextSize calculates the total amount of memory needed by a context, while GrPlaneSize calculates the size of a bitplane in the context. The function:

```
GrContext *GrCreateContext(int w,int h,char far *memory[4],GrContext *where);
```

can be used to create a new context in system memory. The NULL pointer is also accepted as the value of the memory and where arguments, in this case the library allocates the necessary amount of memory internally. It is a general convention in the library that functions returning pointers to any GRX specific data structure have a last argument (most of the time named where in the prototypes) which can be used to pass the address of the data structure which should be filled with the result. If this where pointer has the value of NULL, then the library allocates space for the data structure internally.

The memory argument is really a 4 pointer array, each pointer must point to space to handle GrPlaneSize(w,h) bytes, really only GrNumPlanes() pointers must be malloced, the rest can be NULL. Nevertheless the normal use (see below) is

```
gc = GrCreateContext(w,h,NULL,NULL);
```

so yo don't need to care about.

The function:

```
GrContext *GrCreateSubContext(int x1,int y1,int x2,int y2,
                              const GrContext *parent,GrContext *where);
```

creates a new sub-context which maps to a part of an existing context. The coordinate arguments (x1 through y2) are interpreted relative to the parent context's limits. Pixel addressing is zero-based even in sub-contexts, i.e. the address of the top left pixel is (0,0) even in a sub-context which has been mapped onto the interior of its parent context.

Sub-contexts can be resized, but not their parents (i.e. anything returned by GrCreateContext or set up by GrSetMode cannot be resized – because this could lead to irrecoverable "loss" of drawing memory. The following function can be used for this purpose:

```
void GrResizeSubContext(GrContext *context,int x1,int y1,int x2,int y2);
```

The current context structure is stored in a static location in the library. (For efficiency reasons – it is used quite frequently, and this way no pointer dereferencing is necessary.) The context stores all relevant information about the video organization, coordinate limits, etc... The current context can be set with the:

```
void GrSetContext(const GrContext *context);
```

function. This function will reset the current context to the full graphics screen if it is passed the NULL pointer as argument. The value of the current context can be saved into a GrContext structure pointed to by where using:

```
GrContext *GrSaveContext(GrContext *where);
```

(Again, if where is NULL, the library allocates the space.) The next two functions:

```
const GrContext *GrCurrentContext(void);
const GrContext *GrScreenContext(void);
```

return the current context and the screen context respectively. Contexts can be destroyed with:

```
void GrDestroyContext(GrContext *context);
```

This function will free the memory occupied by the context only if it was allocated originally by the library. The next three functions set up and query the clipping limits associated with the current context:

```
void GrSetClipBox(int x1,int y1,int x2,int y2);
void GrGetClipBox(int *x1p,int *y1p,int *x2p,int *y2p);
void GrResetClipBox(void);
```

GrResetClipBox sets the clipping limits to the limits of context. These are the limits set up initially when a context is created. There are three similar functions to sets/gets the clipping limits of any context:

```
void  GrSetClipBoxC(GrContext *c,int x1,int y1,int x2,int y2);
void  GrGetClipBoxC(const GrContext *c,int *x1p,int *y1p,int *x2p,int *y2p);
void  GrResetClipBoxC(GrContext *c);
```

The limits of the current context can be obtained using the following functions:

```
int GrMaxX(void);
int GrMaxY(void);
int GrSizeX(void);
int GrSizeY(void);
```

The Max functions return the biggest valid coordinate, while the Size functions return a value one higher. The limits of the graphics screen (regardless of the current context) can be obtained with:

```
int GrScreenX(void);
int GrScreenY(void);
```

If you had set a virtual screen (using a custom graphics mode), the limits of the virtual screen can be fetched with:

```
int GrVirtualX(void);
int GrVirtualY(void);
```

The routine:

```
int GrScreenIsVirtual(void);
```

returns non zero if a virtual screen is set. The rectangle showed in the real screen can be set with:

```
int GrSetViewport(int xpos,int ypos);
```

and the current viewport position can be obtained by:

```
int GrViewportX(void);
int GrViewportY(void);
```

Context use

Here is a example of normal context use:

```
GrContext *grc;

if( (grc = GrCreateContext( w,h,NULL,NULL )) == NULL ){
  ...process the error
  }
else {
  GrSetContext( grc );
  ...do some drawing
  ...and probably bitblt to the screen context
  GrSetContext( NULL ); /* the screen context! */
  GrDestroyContext( grc );
  }
```

But if you have a GrContext variable (not a pointer) you want to use (probably because is static to some routines) you can do:

```
static GrContext grc; /* not a pointer!! */

if( GrCreateContext( w,h,NULL,&grc )) == NULL ) {
  ...process the error
 }
else {
  GrSetContext( &grc );
  ...do some drawing
  ...and probably bitblt to the screen context
  GrSetContext( NULL ); /* the screen context! */
  GrDestroyContext( &grc );
  }
```

Note that GrDestoryContext knows if grc was automatically malloced or not!!

Only if you don't want GrCreateContext use malloc at all, you must allocate the memory buffers and pass it to GrCreateContext.

Using GrCreateSubContext is the same, except it doesn't need the buffer, because it uses the parent buffer.

See the see [winclip.c], page 143 and see [wintest.c], page 144 examples.

Color management

GRX defines the type GrColor for color variables. GrColor it's a 32 bits integer. The 8 left bits are reserved for the write mode (see below). The 24 bits right are the color value.

The library supports two models for color management. In the 'indirect' (or color table) model, color values are indices to a color table. The color table slots will be allocated with the highest resolution supported by the hardware (EGA: 2 bits, VGA: 6 bits) with respect to the component color intensities. In the 'direct' (or RGB) model, color values map directly into component color intensities with non-overlapping bitfields of the color index representing the component colors.

Color table model is supported until 256 color modes. The RGB model is supported in 256 color and up color modes.

In RGB model the color index map to component color intensities depend on the video mode set, so it can't be assumed the component color bitfields (but if you are curious check the GrColorInfo global structure in grx20.h).

After the first GrSetMode call two colors are always defined: black and white. The color values of these two colors are returned by the functions:

```
GrColor GrBlack(void);
GrColor GrWhite(void);
```

GrBlack() is guaranteed to be 0.

The library supports five write modes (a write mode descibes the operation between the actual bit color and the one to be set): write, XOR, logical OR, logical AND and IMAGE. These can be selected with OR-ing the color value with one of the following constants declared in grx20.h :

```
#define GrWRITE       0UL            /* write color */
#define GrXOR         0x01000000UL   /* to "XOR" any color to the screen */
#define GrOR          0x02000000UL   /* to "OR" to the screen */
#define GrAND         0x03000000UL   /* to "AND" to the screen */
#define GrIMAGE       0x04000000UL   /* blit: write, except given color */
```

The GrIMAGE write mode only works with the bitblt function. By convention, the no-op color is obtained by combining color value 0 (black) with the XOR operation. This no-op color has been defined in grx20.h as:

```
#define GrNOCOLOR     (GrXOR | 0)    /* GrNOCOLOR is used for "no" color */
```

The write mode part and the color value part of a GrColor variable can be obtained OR-ing it with one of the following constants declared in grx20.h:

```
#define GrCVALUEMASK  0x00ffffffUL   /* color value mask */
#define GrCMODEMASK   0xff000000UL   /* color operation mask */
```

The number of colors in the current graphics mode is returned by the:

```
GrColor GrNumColors(void);
```

function, while the number of unused, available color can be obtained by calling:

```
GrColor GrNumFreeColors(void);
```

Colors can be allocated with the:

```
GrColor GrAllocColor(int r,int g,int b);
GrColor GrAllocColor2(long hcolor);
```

functions (component intensities can range from 0 to 255, hcolor must be in 0xRRGGBB format), or with the:

```
GrColor GrAllocCell(void);
```

function. In the second case the component intensities of the returned color can be set with:

```
void GrSetColor(GrColor color,int r,int g,int b);
```

In the color table model both Alloc functions return GrNOCOLOR if there are no more free colors available. In the RGB model GrNumFreeColors returns 0 and GrAllocCell always returns GrNOCOLOR, as colors returned by GrAllocCell are meant to be changed – what is not supposed to be done in RGB mode. Also note that GrAllocColor operates much more efficiently in RGB mode, and that it never returns GrNOCOLOR in this case.

Color table entries can be freed (when not in RGB mode) by calling:

```
void GrFreeColor(GrColor color);
```

The component intensities of any color can be queried using one of this function:

```
void GrQueryColor(GrColor c,int *r,int *g,int *b);
void GrQueryColor2(GrColor c,long *hcolor);
```

Initially the color system is in color table (indirect) model if there are 256 or less colors. 256 color modes can be put into the RGB model by calling:

```
void GrSetRGBcolorMode(void);
```

The color system can be reset (i.e. put back into color table model if possible, all colors freed except for black and white) by calling:

```
void GrResetColors(void);
```

The function:

```
void GrRefreshColors(void);
```

reloads the currently allocated color values into the video hardware. This function is not needed in typical applications, unless the display adapter is programmed directly by the application.

This functions:

```
GrColor GrAllocColorID(int r,int g,int b);
GrColor GrAllocColor2ID(long hcolor);
void GrQueryColorID(GrColor c,int *r,int *g,int *b);
void GrQueryColor2ID(GrColor c,long *hcolor);
```

are inlined versions (except if you compile GRX with GRX_SKIP_INLINES defined) to be used in the RGB model (in the color table model they call the normal routines).

See the see [rgbtest.c], page 116 and see [colorops.c], page 48 examples.

Portable use of a few colors

People that only want to use a few colors find the GRX color handling a bit confusing, but it gives the power to manage a lot of color deeps and two color models. Here are some guidelines to easily use the famous 16 ega colors in GRX programs. We need this GRX function:

```
GrColor *GrAllocEgaColors(void);
```

it returns a 16 GrColor array with the 16 ega colors alloced (really it's a trivial function, read the source src/setup/colorega.c). We can use a construction like that:

First, in your C code make a global pointer, and init it after set the graphics mode:

```
GrColor *egacolors;
....
int your_setup_function( ... )
{
  ...
  GrSetMode( ... )
  ...
  egacolors = GrAllocEgaColors();
  ...
}
```

Next, add this to your main include file:

```
extern GrColor *egacolors;
#define BLACK        egacolors[0]
#define BLUE         egacolors[1]
#define GREEN        egacolors[2]
#define CYAN         egacolors[3]
#define RED          egacolors[4]
#define MAGENTA      egacolors[5]
#define BROWN        egacolors[6]
#define LIGHTGRAY    egacolors[7]
#define DARKGRAY     egacolors[8]
#define LIGHTBLUE    egacolors[9]
#define LIGHTGREEN   egacolors[10]
#define LIGHTCYAN    egacolors[11]
#define LIGHTRED     egacolors[12]
#define LIGHTMAGENTA egacolors[13]
#define YELLOW       egacolors[14]
#define WHITE        egacolors[15]
```

Now you can use the defined colors in your code. Note that if you are in color table model in a 16 color mode, you have exhausted the color table. Note too that this don't work to initialize static variables with a color, because egacolors is not initialized.

Graphics primitives

The screen, the current context or the current clip box can be cleared (i.e. set to a desired background color) by using one of the following three functions:

```
void GrClearScreen(GrColor bg);
void GrClearcontext(GrColor bg);
void GrClearClipBox(GrColor bg);
```

Thanks to the special GrColor definition, you can do more than simple clear with this functions, by example with:

```
GrClearScreen( GrWhite()|GrXOR );
```

the graphics screen is negativized, do it again and the screen is restored.

The following line drawing graphics primitives are supported by the library:

```
void GrPlot(int x,int y,GrColor c);
void GrLine(int x1,int y1,int x2,int y2,GrColor c);
void GrHLine(int x1,int x2,int y,GrColor c);
void GrVLine(int x,int y1,int y2,GrColor c);
void GrBox(int x1,int y1,int x2,int y2,GrColor c);
void GrCircle(int xc,int yc,int r,GrColor c);
void GrEllipse(int xc,int yc,int xa,int ya,GrColor c);
void GrCircleArc(int xc,int yc,int r,int start,int end,int style,GrColor c);
void GrEllipseArc(int xc,int yc,int xa,int ya,
                  int start,int end,int style,GrColor c);
void GrPolyLine(int numpts,int points[][2],GrColor c);
void GrPolygon(int numpts,int points[][2],GrColor c);
```

All primitives operate on the current graphics context. The last argument of these functions is always the color to use for the drawing. The HLine and VLine primitives are for drawing horizontal and vertical lines. They have been included in the library because they are more efficient than the general line drawing provided by GrLine. The ellipse primitives can only draw ellipses with their major axis parallel with either the X or Y coordinate axis. They take the half X and Y axis length in the xa and ya arguments. The arc (circle and ellipse) drawing functions take the start and end angles in tenths of degrees (i.e. meaningful range: 0 ... 3600). The angles are interpreted counter-clockwise starting from the positive X axis. The style argument can be one of this defines from grx20.h:

```
#define GR_ARC_STYLE_OPEN       0
#define GR_ARC_STYLE_CLOSE1     1
#define GR_ARC_STYLE_CLOSE2     2
```

GR_ARC_STYLE_OPEN draws only the arc, GR_ARC_STYLE_CLOSE1 closes the arc with a line between his start and end point, GR_ARC_STYLE_CLOSE1 draws the typical cake slice. This routine:

```
void GrLastArcCoords(int *xs,int *ys,int *xe,int *ye,int *xc,int *yc);
```

can be used to retrieve the start, end, and center points used by the last arc drawing functions.

See the see [circtest.c], page 45 and see [arctest.c], page 35 examples.

The polyline and polygon primitives take the address of an n by 2 coordinate array. The X values should be stored in the elements with 0 second index, and the Y values in the elements with a second index value of 1. Coordinate arrays passed to the polygon primitive can either contain or omit the closing edge of the polygon – the primitive will append it to the list if it is missing.

See the see [polytest.c], page 112 example.

Because calculating the arc points it's a very time consuming operation, there are two functions to pre-calculate the points, that can be used next with polyline and polygon primitives:

```
int  GrGenerateEllipse(int xc,int yc,int xa,int ya,
                       int points[GR_MAX_ELLIPSE_POINTS][2]);
int  GrGenerateEllipseArc(int xc,int yc,int xa,int ya,int start,int end,
                          int points[GR_MAX_ELLIPSE_POINTS][2]);
```

The following filled primitives are available:

```
void GrFilledBox(int x1,int y1,int x2,int y2,GrColor c);
void GrFramedBox(int x1,int y1,int x2,int y2,int wdt,const GrFBoxColors *c);
void GrFilledCircle(int xc,int yc,int r,GrColor c);
void GrFilledEllipse(int xc,int yc,int xa,int ya,GrColor c);
void GrFilledCircleArc(int xc,int yc,int r,
                       int start,int end,int style,GrColor c);
void GrFilledEllipseArc(int xc,int yc,int xa,int ya,
                        int start,int end,int style,GrColor c);
void GrFilledPolygon(int numpts,int points[][2],GrColor c);
void GrFilledConvexPolygon(int numpts,int points[][2],GrColor c);
```

Similarly to the line drawing, all of the above primitives operate on the current graphics context. The GrFramedBox primitive can be used to draw motif-like shaded boxes and "ordinary" framed boxes as well. The x1 through y2 coordinates specify the interior of the box, the border is outside this area, wdt pixels wide. The primitive uses five different colors for the interior and four borders of the box which are specified in the GrFBoxColors structure:

```
typedef struct {
  GrColor fbx_intcolor;
  GrColor fbx_topcolor;
  GrColor fbx_rightcolor;
  GrColor fbx_bottomcolor;
  GrColor fbx_leftcolor;
} GrFBoxColors;
```

The GrFilledConvexPolygon primitive can be used to fill convex polygons. It can also be used to fill some concave polygons whose boundaries do not intersect any horizontal scan line more than twice. All other concave polygons have to be filled with the (somewhat less efficient) GrFilledPolygon primitive. This primitive can also be used to fill several disjoint nonoverlapping polygons in a single operation.

The function:

```
void GrFloodFill(int x, int y, GrColor border, GrColor c);
```

flood-fills the area bounded by the color border using x, y like the starting point.

The current color value of any pixel in the current context can be obtained with:

```
GrColor GrPixel(int x,int y);
```

and:

```
GrColor GrPixelC(GrContext *c,int x,int y);
```

do the same for any context.

Rectangular areas can be transferred within a context or between contexts by calling:

```
void GrBitBlt(GrContext *dest,int x,int y,GrContext *source,
              int x1,int y1,int x2,int y2,GrColor op);
```

x, y is the position in the destination context, and x1, y1, x2, y2 the area from the source context to be transfered. The op argument should be one of supported color write modes (GrWRITE, GrXOR, GrOR, GrAND, GrIMAGE), it will control how the pixels from the source context are combined with the pixels in the destination context (the GrIMAGE op must be ored with the color value to be handled as transparent). If either the source or the destination context argument is the NULL pointer then the current context is used for that argument.

See the see [blittest.c], page 38 example.

A efficient form to get/put pixels from/to a context can be achieved using the next functions:

```
const GrColor *GrGetScanline(int x1,int x2,int yy);
const GrColor *GrGetScanlineC(GrContext *ctx,int x1,int x2,int yy);
void GrPutScanline(int x1,int x2,int yy,const GrColor *c, GrColor op);
```

The Get functions return a pointer to a static GrColor pixel array (or NULL if they fail) with the color values of a row (yy) segment (x1 to x2). GrGetScanline uses the current context. GrGestScanlineC uses the context ctx (that can be NULL to refer to the current context). Note that the output is only valid until the next GRX call.

GrPutScanline puts the GrColor pixel array c on the yy row segmet defined by x1 to x2 in the current context using the op operation. op can be any of GrWRITE, GrXOR, GrOR, GrAND or GrIMAGE. Data in c must fit GrCVALUEMASK otherwise the results are implementation dependend. So you can't supply operation code with the pixel data!.

Non-clipping graphics primitives

There is a non-clipping version of some of the elementary primitives. These are somewhat more efficient than the regular versions. These are to be used only in situations when it is absolutely certain that no drawing will be performed beyond the boundaries of the current context. Otherwise the program will almost certainly crash! The reason for including these functions is that they are somewhat more efficient than the regular, clipping versions. ALSO NOTE: These function do not check for conflicts with the mouse cursor. (See the explanation about the mouse cursor handling later in this document.) The list of the supported non-clipping primitives:

```
void GrPlotNC(int x,int y,GrColor c);
void GrLineNC(int x1,int y1,int x2,int y2,GrColor c);
void GrHLineNC(int x1,int x2,int y,GrColor c);
void GrVLineNC(int x,int y1,int y2,GrColor c);
void GrBoxNC(int x1,int y1,int x2,int y2,GrColor c);
void GrFilledBoxNC(int x1,int y1,int x2,int y2,GrColor c);
void GrFramedBoxNC(int x1,int y1,int x2,int y2,int wdt,const GrFBoxCol-
ors *c);
void grbitbltNC(GrContext *dst,int x,int y,GrContext *src,
                int x1,int y1,int x2,int y2,GrColor op);
GrColor GrPixelNC(int x,int y);
GrColor GrPixelCNC(GrContext *c,int x,int y);
```

Customized line drawing

The basic line drawing graphics primitives described previously always draw continuous lines which are one pixel wide. There is another group of line drawing functions which can be used to draw wide and/or patterned lines. These functions have similar parameter passing conventions as the basic ones with one difference: instead of the color value a pointer to a structure of type GrLineOption has to be passed to them. The definition of the GrLineOption structure:

```
typedef struct {
  GrColor lno_color;              /* color used to draw line */
  int     lno_width;              /* width of the line */
  int     lno_pattlen;            /* length of the dash pattern */
  unsigned char *lno_dashpat;     /* draw/nodraw pattern */
} GrLineOption;
```

The lno_pattlen structure element should be equal to the number of alternating draw – no draw section length values in the array pointed to by the lno_dashpat element. The dash pattern array is assumed to begin with a drawn section. If the pattern length is equal to zero a continuous line is drawn.

Example, a white line 3 bits wide (thick) and pattern 6 bits draw, 4 bits nodraw:

```
GrLineOption mylineop;
...
mylineop.lno_color = GrWhite();
mylineop.lno_width = 3;
mylineop.lno_pattlen = 2;
mylineop.lno_dashpat = "\x06\x04";
```

The available custom line drawing primitives:

```
void GrCustomLine(int x1,int y1,int x2,int y2,const GrLineOption *o);
void GrCustomBox(int x1,int y1,int x2,int y2,const GrLineOption *o);
void GrCustomCircle(int xc,int yc,int r,const GrLineOption *o);
void GrCustomEllipse(int xc,int yc,int xa,int ya,const GrLineOption *o);
void GrCustomCircleArc(int xc,int yc,int r,
                       int start,int end,int style,const GrLineOption *o);
void GrCustomEllipseArc(int xc,int yc,int xa,int ya,
                        int start,int end,int style,const GrLineOption *o);
void GrCustomPolyLine(int numpts,int points[][2],const GrLineOption *o);
void GrCustomPolygon(int numpts,int points[][2],const GrLineOption *o);
```

See the see [linetest.c], page 100 example.

Pattern filled graphics primitives

The library also supports a pattern filled version of the basic filled primitives described above. These functions have similar parameter passing conventions as the basic ones with one difference: instead of the color value a pointer to an union of type 'GrPattern' has to be passed to them. The GrPattern union can contain either a bitmap or a pixmap fill pattern. The first integer slot in the union determines which type it is. Bitmap fill patterns are rectangular arrays of bits, each set bit representing the foreground color of the fill operation, and each zero bit representing the background. Both the foreground and background colors can be combined with any of the supported logical operations. Bitmap fill patterns have one restriction: their width must be eight pixels. Pixmap fill patterns are very similar to contexts. The relevant structure declarations (from grx20.h):

```
/*
 * BITMAP: a mode independent way to specify a fill pattern of two
 *   colors. It is always 8 pixels wide (1 byte per scan line), its
 *   height is user-defined. SET THE TYPE FLAG TO ZERO!!!
 */
typedef struct _GR_bitmap {
  int     bmp_ispixmap;          /* type flag for pattern union */
  int     bmp_height;            /* bitmap height */
  char   *bmp_data;              /* pointer to the bit pattern */
  GrColor bmp_fgcolor;           /* foreground color for fill */
  GrColor bmp_bgcolor;           /* background color for fill */
  int     bmp_memflags;          /* set if dynamically allocated */
} GrBitmap;

/*
 * PIXMAP: a fill pattern stored in a layout identical to the video RAM
 *   for filling using 'bitblt'-s. It is mode dependent, typically one
 *   of the library functions is used to build it. KEEP THE TYPE FLAG
 *   NONZERO!!!
 */
typedef struct _GR_pixmap {
  int     pxp_ispixmap;          /* type flag for pattern union */
  int     pxp_width;             /* pixmap width (in pixels)  */
  int     pxp_height;            /* pixmap height (in pixels) */
  GrColor pxp_oper;              /* bitblt mode (SET, OR, XOR, AND, IM-
AGE) */
  struct _GR_frame pxp_source;   /* source context for fill */
} GrPixmap;

/*
 * Fill pattern union -- can either be a bitmap or a pixmap
 */
typedef union _GR_pattern {
  int      gp_ispixmap;          /* nonzero for pixmaps */
  GrBitmap gp_bitmap;            /* fill bitmap */
  GrPixmap gp_pixmap;            /* fill pixmap */
} GrPattern;
```

This define group (from grx20.h) help to acces the GrPattern menbers:

```
#define gp_bmp_data                         gp_bitmap.bmp_data
#define gp_bmp_height                       gp_bitmap.bmp_height
#define gp_bmp_fgcolor                      gp_bitmap.bmp_fgcolor
#define gp_bmp_bgcolor                      gp_bitmap.bmp_bgcolor

#define gp_pxp_width                        gp_pixmap.pxp_width
#define gp_pxp_height                       gp_pixmap.pxp_height
#define gp_pxp_oper                         gp_pixmap.pxp_oper
#define gp_pxp_source                       gp_pixmap.pxp_source
```

Bitmap patterns can be easily built from initialized character arrays and static structures by the C compiler, thus no special support is included in the library for creating them. The only action required from the application program might be changing the foreground and background colors as needed. Pixmap patterns are more difficult to build as they replicate the layout of the video memory which changes for different video modes. For this reason the library provides three functions to create pixmap patterns in a mode-independent way:

```
GrPattern *GrBuildPixmap(const char *pixels,int w,int h,const GrColorTableP colors);
GrPattern *GrBuildPixmapFromBits(const char *bits,int w,int h,
                                 GrColor fgc,GrColor bgc);
GrPattern *GrConvertToPixmap(GrContext *src);
```

GrBuildPixmap build a pixmap from a two dimensional (w by h) array of characters. The elements in this array are used as indices into the color table specified with the argument colors. (This means that pixmaps created this way can use at most 256 colors.) The color table pointer:

```
typedef GrColor *GrColorTableP;
```

should point to an array of integers with the first element being the number of colors in the table and the color values themselves starting with the second element. NOTE: any color modifiers (GrXOR, GrOR, GrAND) OR-ed to the elements of the color table are ignored.

The GrBuildPixmapFromBits function builds a pixmap fill pattern from bitmap data. It is useful if the width of the bitmap pattern is not eight as such bitmap patterns can not be used to build a GrBitmap structure.

The GrConvertToPixmap function converts a graphics context to a pixmap fill pattern. It is useful when the pattern can be created with graphics drawing operations. NOTE: the pixmap pattern and the original context share the drawing RAM, thus if the context is redrawn the fill pattern changes as well. Fill patterns which were built by library routines can be destroyed when no longer needed (i.e. the space occupied by them can be freed) by calling:

```
void GrDestroyPattern(GrPattern *p);
```

NOTE: when pixmap fill patterns converted from contexts are destroyed, the drawing RAM is not freed. It is freed when the original context is destroyed. Fill patterns built by the application have to be destroyed by the application as well (if this is needed).

The list of supported pattern filled graphics primitives is shown below. These functions are very similar to their solid filled counterparts, only their last argument is different:

```
void GrPatternFilledPlot(int x,int y,GrPattern *p);
void GrPatternFilledLine(int x1,int y1,int x2,int y2,GrPattern *p);
void GrPatternFilledBox(int x1,int y1,int x2,int y2,GrPattern *p);
void GrPatternFilledCircle(int xc,int yc,int r,GrPattern *p);
void GrPatternFilledEllipse(int xc,int yc,int xa,int ya,GrPattern *p);
void GrPatternFilledCircleArc(int xc,int yc,int r,int start,int end,
                              int style,GrPattern *p);
void GrPatternFilledEllipseArc(int xc,int yc,int xa,int ya,int start,int end,
                               int style,GrPattern *p);
void GrPatternFilledConvexPolygon(int numpts,int points[][2],GrPattern *p);
void GrPatternFilledPolygon(int numpts,int points[][2],GrPattern *p);
void GrPatternFloodFill(int x, int y, GrColor border, GrPattern *p);
```

Strictly speaking the plot and line functions in the above group are not filled, but they have been included here for convenience.

Patterned line drawing

The custom line drawing functions introduced above also have a version when the drawn sections can be filled with a (pixmap or bitmap) fill pattern. To achieve this these functions must be passed both a custom line drawing option (GrLineOption structure) and a fill pattern (GrPattern union). These two have been combined into the GrLinePattern structure:

```
typedef struct {
  GrPattern     *lnp_pattern;    /* fill pattern */
  GrLineOption  *lnp_option;     /* width + dash pattern */
} GrLinePattern;
```

All patterned line drawing functions take a pointer to this structure as their last argument. The list of available functions:

```
void GrPatternedLine(int x1,int y1,int x2,int y2,GrLinePattern *lp);
void GrPatternedBox(int x1,int y1,int x2,int y2,GrLinePattern *lp);
void GrPatternedCircle(int xc,int yc,int r,GrLinePattern *lp);
void GrPatternedEllipse(int xc,int yc,int xa,int ya,GrLinePattern *lp);
void GrPatternedCircleArc(int xc,int yc,int r,int start,int end,
                          int style,GrLinePattern *lp);
void GrPatternedEllipseArc(int xc,int yc,int xa,int ya,int start,int end,
                           int style,GrLinePattern *lp);
void GrPatternedPolyLine(int numpts,int points[][2],GrLinePattern *lp);
void GrPatternedPolygon(int numpts,int points[][2],GrLinePattern *lp);
```

Image manipulation

GRX defines the GrImage type like a GrPixmap synonym:

```
#define GrImage GrPixmap
```

nevertheless the GrImage type enforces the image character of this object, so for compatibility with future GRX versions use the next functions if you need to convert between GrImage and GrPixmap objects:

```
GrImage *GrImageFromPattern(GrPattern *p);
GrPattern *GrPatternFromImage(GrImage *p);
```

the GrImageFromPattern function returns NULL if the GrPattern given is not a GrPixmap.

Like pixmaps patterns images are dependent of the actual video mode set. So the library provides functions to create images in a mode-independent way:

```
GrImage *GrImageBuild(const char *pixels,int w,int h,const GrColorTableP colors);
GrImage *GrImageFromContext(GrContext *c);
```

these functions work like the GrBuildPixmap and GrConvertToPixmap ones. Remember: the image and the original context share the drawing RAM.

There are a number of functions to display all or part of an image in the current context:

```
void GrImageDisplay(int x,int y, GrImage *i);
void GrImageDisplayExt(int x1,int y1,int x2,int y2, GrImage *i);
void GrImageFilledBoxAlign(int xo,int yo,int x1,int y1,int x2,int y2,
                           GrImage *p);
void GrImageHLineAlign(int xo,int yo,int x,int y,int width,GrImage *p);
void GrImagePlotAlign(int xo,int yo,int x,int y,GrImage *p);
```

GrImageDisplay display the whole image using x, y like the upper left corner in the current context. GrImageDisplayExt display as much as it can (repiting the image if necesary) in the rectangle defined by x1, y1 and x2, y2.

GrImageFilledBoxAlign is a most general funtion (really the later two call it) display as much as it can in the defined rectangle using xo, yo like the align point, it is the virtual point in the destination context (it doesn't need to be into the rectangle) with that the upper left image corner is aligned.

GrImageHLineAlign and GrImagePlotAlign display a row segment or a point of the image at x y position using the xo, yo allign point.

The most usefull image funtions are these:

```
GrImage *GrImageInverse(GrImage *p,int flag);
GrImage *GrImageStretch(GrImage *p,int nwidth,int nheight);
```

GrImageInverse creates a new image object, flipping p left-right or top-down as indicated by flag that can be:

```
#define GR_IMAGE_INVERSE_LR  0x01  /* inverse left right */
#define GR_IMAGE_INVERSE_TD  0x02  /* inverse top down */
```

GrImageStretch creates a new image stretching p to nwidth by nheight.

To destroy a image objet when you don't need it any more use:

```
void GrImageDestroy(GrImage *i);
```

See the see [imgtest.c], page 83 example.

Text drawing

The library supports loadable fonts. When in memory they are bit-mapped (i.e. not scalable!) fonts. A driver design allow GRX to load different font formats, the last GRX release come with drivers to load the GRX own font format and the BGI Borland format for all platforms supported, the X11 version can load X11 fonts too.

The GRX distribution come with a font collection in the GRX own format. Some of these fonts were converted from VGA fonts. These fonts have all 256 characters from the PC-437 codepage. Some additional fonts were converted from fonts in the MIT X11 distribution. Most of these are ISO-8859-1 coded. Fonts also have family names. The following font families are included:

Font file name	Family	Description
`pc<W>x<H>[t].fnt`	`pc`	`VGA font, fixed`
`xm<W>x<H>[b][i].fnt`	`X_misc`	`X11, fixed, miscellaneous group`
`char<H>[b][i].fnt`	`char`	`X11, proportional, charter family`
`cour<H>[b][i].fnt`	`cour`	`X11, fixed, courier`
`helve<H>[b][i].fnt`	`helve`	`X11, proportional, helvetica`
`lucb<H>[b][i].fnt`	`lucb`	`X11, proportional, lucida bright`
`lucs<H>[b][i].fnt`	`lucs`	`X11, proportional, lucida sans serif`
`luct<H>[b][i].fnt`	`luct`	`X11, fixed, lucida typewriter`
`ncen<H>[b][i].fnt`	`ncen`	`X11, proportional, new century schoolbook`
`symb<H>.fnt`	`symbol`	`X11, proportional, greek letters, symbols`
`tms<H>[b][i].fnt`	`times`	`X11, proportional, times`

In the font names `<W>` means the font width, `<H>` the font height. Many font families have bold and/or italic variants. The files containing these fonts contain a ’b’ and/or ’i’ character in their name just before the extension. Additionally, the strings "_bold" and/or "_ital" are appended to the font family names. Some of the pc VGA fonts come in thin formats also, these are denoted by a ’t’ in their file names and the string "_thin" in their family names.

The GrFont structure hold a font in memory. A number of ’pc’ fonts are built-in to the library and don’t need to be loaded:

```
extern  GrFont          GrFont_PC6x8;
extern  GrFont          GrFont_PC8x8;
extern  GrFont          GrFont_PC8x14;
extern  GrFont          GrFont_PC8x16;
```

Other fonts must be loaded with the GrLoadFont function. If the font file name starts with any path separator character or character sequence (’:’, ’/’ or ’\’) then it is loaded from the specified directory, otherwise the library try load the font first from the current directory and next from the default font path. The font path can be set up with the GrSetFontPath function. If the font path is not set then the value of the ’GRXFONT’ environment variable is used as the font path. If GrLoadFont is called again with the name of an already loaded font then it will return a pointer to the result of the first loading. Font loading routines return NULL if the font was not found. When not needed any more, fonts can be unloaded (i.e. the storage occupied by them freed) by calling GrUnloadFont.

The prototype declarations for these functions:

```
GrFont *GrLoadFont(char *name);
void GrUnloadFont(GrFont *font);
void GrSetFontPath(char *path_list);
```

Using these functions:

```
GrFont *GrLoadConvertedFont(char *name,int cvt,int w,int h,
                            int minch,int maxch);
```

```
GrFont *GrBuildConvertedFont(const GrFont *from,int cvt,int w,int h,
                             int minch,int maxch);
```

a new font can be generated from a file font or a font in memory, the 'cvt' argument direct the conversion or-ing the desired operations from these defines:

```
/*
 * Font conversion flags for 'GrLoadConvertedFont'. OR them as desired.
 */
#define GR_FONTCVT_NONE         0     /* no conversion */
#define GR_FONTCVT_SKIPCHARS    1     /* load only selected characters */
#define GR_FONTCVT_RESIZE       2     /* resize the font */
#define GR_FONTCVT_ITALICIZE    4     /* tilt font for "italic" look */
#define GR_FONTCVT_BOLDIFY      8     /* make a "bold"(er) font  */
#define GR_FONTCVT_FIXIFY       16    /* convert prop. font to fixed wdt */
#define GR_FONTCVT_PROPORTION   32    /* convert fixed font to prop. wdt */
```

GR_FONTCVT_SKIPCHARS needs 'minch' and 'maxch' arguments.

GR_FONTCVT_RESIZE needs 'w' and 'h' arguments.

The function:

```
void GrDumpFnaFont(const GrFont *f, char *fileName);
```

writes a font to an ascii font file, so it can be quickly edited with a text editor. For a description of the ascii font format, see the fna.txt file.

The function:

```
void GrDumpFont(const GrFont *f,char *CsymbolName,char *fileName);
```

writes a font to a C source code file, so it can be compiled and linked with a user program. GrDumpFont would not normally be used in a release program because its purpose is to produce source code. When the source code is compiled and linked into a program distributing the font file with the program in not necessary, avoiding the possibility of the font file being deleted or corrupted.

You can use the premade fnt2c.c program (see the source, it's so simple) to dump a selected font to source code, by example:

"fnt2c helv15 myhelv15 myhelv15.c"

Next, if this line is included in your main include file:

```
extern GrFont myhelv15
```

and "myhelv15.c" compiled and linked with your project, you can use 'myhelv15' in every place a GrFont is required.

This simple function:

```
void GrTextXY(int x,int y,char *text,GrColor fg,GrColor bg);
```

draw text in the current context in the standard direction, using the GrDefaultFont (mapped in the grx20.h file to the GrFont_PC8x14 font) with x, y like the upper left corner and the foreground and background colors given (note that bg equal to GrNOCOLOR make the background transparent).

For other functions the GrTextOption structure specifies how to draw a character string:

```
typedef struct _GR_textOption {      /* text drawing option structure */
  struct _GR_font     *txo_font;      /* font to be used */
  union  _GR_textColor txo_fgcolor;   /* foreground color */
  union  _GR_textColor txo_bgcolor;   /* background color */
  char    txo_chrtype;                /* character type (see above) */
  char    txo_direct;                 /* direction (see above) */
  char    txo_xalign;                 /* X alignment (see above) */
  char    txo_yalign;                 /* Y alignment (see above) */
} GrTextOption;

typedef union _GR_textColor {        /* text color union */
  GrColor       v;                    /* color value for "direct" text */
  GrColorTableP p;                    /* color table for attribute text */
} GrTextColor;
```

The text can be rotated in increments of 90 degrees (txo_direct), alignments can be set in both directions (txo_xalign and txo_yalign), and separate fore and background colors can be specified. The accepted text direction values:

```
#define GR_TEXT_RIGHT           0       /* normal */
#define GR_TEXT_DOWN            1       /* downward */
#define GR_TEXT_LEFT            2       /* upside down, right to left */
#define GR_TEXT_UP              3       /* upward */
#define GR_TEXT_DEFAULT         GR_TEXT_RIGHT
```

The accepted horizontal and vertical alignment option values:

```
#define GR_ALIGN_LEFT           0       /* X only */
#define GR_ALIGN_TOP            0       /* Y only */
#define GR_ALIGN_CENTER         1       /* X, Y   */
#define GR_ALIGN_RIGHT          2       /* X only */
#define GR_ALIGN_BOTTOM         2       /* Y only */
#define GR_ALIGN_BASELINE       3       /* Y only */
#define GR_ALIGN_DEFAULT        GR_ALIGN_LEFT
```

Text strings can be of three different types: one character per byte (i.e. the usual C character string, this is the default), one character per 16-bit word (suitable for fonts with a large number of characters), and a PC-style character-attribute pair. In the last case the GrTextOption structure must contain a pointer to a color table of size 16 (fg color bits in attrib) or 8 (bg color bits). (The color table format is explained in more detail in the previous section explaining the methods to build fill patterns.) The supported text types:

```
#define GR_BYTE_TEXT            0       /* one byte per character */
#define GR_WORD_TEXT            1       /* two bytes per character */
#define GR_ATTR_TEXT            2       /* chr w/ PC style attribute byte */
```

The PC-style attribute text uses the same layout (first byte: character, second: attributes) and bitfields as the text mode screen on the PC. The only difference is that the 'blink' bit is not supported (it would be very time consuming – the PC text mode does it with hardware support). This bit is used instead to control the underlined display of characters. For convenience the following attribute manipulation macros have been declared in grx20.h:

```
#define GR_BUILD_ATTR(fg,bg,ul) \
        (((fg) & 15) | (((bg) & 7) << 4) | ((ul) ? 128 : 0))
#define GR_ATTR_FGCOLOR(attr)   (((attr)     ) &  15)
#define GR_ATTR_BGCOLOR(attr)   (((attr) >> 4) &   7)
#define GR_ATTR_UNDERLINE(attr) (((attr)     ) & 128)
```

Text strings of the types GR_BYTE_TEXT and GR_WORD_TEXT can also be drawn underlined. This is controlled by OR-ing the constant GR_UNDERLINE_TEXT to the foreground color value:

```
#define GR_UNDERLINE_TEXT       (GrXOR << 4)
```

After the application initializes a text option structure with the desired values it can call one of the following two text drawing functions:

```
void GrDrawChar(int chr,int x,int y,const GrTextOption *opt);
void GrDrawString(void *text,int length,int x,int y,const GrTextOption *opt);
```

NOTE: text drawing is fastest when it is drawn in the 'normal' direction, and the character does not have to be clipped. It this case the library can use the appropriate low-level video RAM access routine, while in any other case the text is drawn pixel-by-pixel by the higher-level code.

There are pattern filed versions too:

```
void GrPatternDrawChar(int chr,int x,int y,const GrTextOption *opt,GrPattern *p);
void GrPatternDrawString(void *text,int length,int x,int y,const GrTex-
tOption *opt,
                         GrPattern *p);
void GrPatternDrawStringExt(void *text,int length,int x,int y,
                            const GrTextOption *opt,GrPattern *p);
```

The size of a font, a character or a text string can be obtained by calling one of the following functions. These functions also take into consideration the text direction specified in the text option structure passed to them.

```
int  GrFontCharPresent(const GrFont *font,int chr);
int  GrFontCharWidth(const GrFont *font,int chr);
int  GrFontCharHeight(const GrFont *font,int chr);
int  GrFontCharBmpRowSize(const GrFont *font,int chr);
int  GrFontCharBitmapSize(const GrFont *font,int chr);
int  GrFontStringWidth(const GrFont *font,void *text,int len,int type);
int  GrFontStringHeight(const GrFont *font,void *text,int len,int type);
int  GrProportionalTextWidth(const GrFont *font,void *text,int len,int type);
int  GrCharWidth(int chr,const GrTextOption *opt);
int  GrCharHeight(int chr,const GrTextOption *opt);
void GrCharSize(int chr,const GrTextOption *opt,int *w,int *h);
int  GrStringWidth(void *text,int length,const GrTextOption *opt);
int  GrStringHeight(void *text,int length,const GrTextOption *opt);
void GrStringSize(void *text,int length,const GrTextOption *opt,int *w,int *h);
```

The GrTextRegion structure and its associated functions can be used to implement a fast (as much as possible in graphics modes) rectangular text window using a fixed font. Clipping for such windows is done in character size increments instead of pixels (i.e. no partial characters are drawn). Only fixed fonts can be used in their natural size. GrDumpText will cache the code of the drawn characters in the buffer pointed to by the 'backup' slot (if it is non-NULL) and will draw a character only if the previously drawn character in that grid element is different.

This can speed up text scrolling significantly in graphics modes. The supported text types are the same as above.

```
typedef struct {                     /* fixed font text window desc. */
  struct _GR_font     *txr_font;      /* font to be used */
  union  _GR_textColor txr_fgcolor;   /* foreground color */
  union  _GR_textColor txr_bgcolor;   /* background color */
  void   *txr_buffer;                 /* pointer to text buffer */
  void   *txr_backup;                 /* optional backup buffer */
  int     txr_width;                  /* width of area in chars */
  int     txr_height;                 /* height of area in chars */
  int     txr_lineoffset;             /* offset in buffer(s) between rows */
  int     txr_xpos;                   /* upper left corner X coordinate */
  int     txr_ypos;                   /* upper left corner Y coordinate */
  char    txr_chrtype;                /* character type (see above) */
} GrTextRegion;

void GrDumpChar(int chr,int col,int row,const GrTextRegion *r);
void GrDumpText(int col,int row,int wdt,int hgt,const GrTextRegion *r);
void GrDumpTextRegion(const GrTextRegion *r);
```

The GrDumpTextRegion function outputs the whole text region, while GrDumpText draws only a user-specified part of it. GrDumpChar updates the character in the buffer at the specified location with the new character passed to it as argument and then draws the new character on the screen as well. With these functions you can simulate a text mode window, write chars directly to the txr_buffer and call GrDumpTextRegion when you want to update the window (or GrDumpText if you know the area to update is small).

See the see [fonttest.c], page 69 example.

Drawing in user coordinates

There is a second set of the graphics primitives which operates in user coordinates. Every context has a user to screen coordinate mapping associated with it. An application specifies the user window by calling the GrSetUserWindow function.

```
void GrSetUserWindow(int x1,int y1,int x2,int y2);
```

A call to this function it in fact specifies the virtual coordinate limits which will be mapped onto the current context regardless of the size of the context. For example, the call:

```
GrSetUserWindow(0,0,11999,8999);
```

tells the library that the program will perform its drawing operations in a coordinate system X:0...11999 (width = 12000) and Y:0...8999 (height = 9000). This coordinate range will be mapped onto the total area of the current context. The virtual coordinate system can also be shifted. For example:

```
GrSetUserWindow(5000,2000,16999,10999);
```

The user coordinates can even be used to turn the usual left-handed coordinate system (0:0 corresponds to the upper left corner) to a right handed one (0:0 corresponds to the bottom left corner) by calling:

```
GrSetUserWindow(0,8999,11999,0);
```

The library also provides three utility functions for the query of the current user coordinate limits and for converting user coordinates to screen coordinates and vice versa.

```
void GrGetUserWindow(int *x1,int *y1,int *x2,int *y2);
void GrGetScreenCoord(int *x,int *y);
void GrGetUserCoord(int *x,int *y);
```

If an application wants to take advantage of the user to screen coordinate mapping it has to use the user coordinate version of the graphics primitives. These have exactly the same parameter passing conventions as their screen coordinate counterparts. NOTE: the user coordinate system is not initialized by the library! The application has to set up its coordinate mapping before calling any of the use coordinate drawing functions – otherwise the program will almost certainly exit (in a quite ungraceful fashion) with a 'division by zero' error. The list of supported user coordinate drawing functions:

```
void GrUsrPlot(int x,int y,GrColor c);
void GrUsrLine(int x1,int y1,int x2,int y2,GrColor c);
void GrUsrHLine(int x1,int x2,int y,GrColor c);
void GrUsrVLine(int x,int y1,int y2,GrColor c);
void GrUsrBox(int x1,int y1,int x2,int y2,GrColor c);
void GrUsrFilledBox(int x1,int y1,int x2,int y2,GrColor c);
void GrUsrFramedBox(int x1,int y1,int x2,int y2,int wdt,GrFBoxColors *c);
void GrUsrCircle(int xc,int yc,int r,GrColor c);
void GrUsrEllipse(int xc,int yc,int xa,int ya,GrColor c);
void GrUsrCircleArc(int xc,int yc,int r,int start,int end,
                    int style,GrColor c);
void GrUsrEllipseArc(int xc,int yc,int xa,int ya,int start,int end,
                     int style,GrColor c);
void GrUsrFilledCircle(int xc,int yc,int r,GrColor c);
void GrUsrFilledEllipse(int xc,int yc,int xa,int ya,GrColor c);
void GrUsrFilledCircleArc(int xc,int yc,int r,int start,int end,
                          int style,GrColor c);
void GrUsrFilledEllipseArc(int xc,int yc,int xa,int ya,int start,int end,
                           int style,GrColor c);
void GrUsrPolyLine(int numpts,int points[][2],GrColor c);
void GrUsrPolygon(int numpts,int points[][2],GrColor c);
void GrUsrFilledConvexPolygon(int numpts,int points[][2],GrColor c);
void GrUsrFilledPolygon(int numpts,int points[][2],GrColor c);
void GrUsrFloodFill(int x, int y, GrColor border, GrColor c);
GrColor GrUsrPixel(int x,int y);
GrColor GrUsrPixelC(GrContext *c,int x,int y);
void GrUsrCustomLine(int x1,int y1,int x2,int y2,const GrLineOption *o);
void GrUsrCustomBox(int x1,int y1,int x2,int y2,const GrLineOption *o);
void GrUsrCustomCircle(int xc,int yc,int r,const GrLineOption *o);
void GrUsrCustomEllipse(int xc,int yc,int xa,int ya,const GrLineOption *o);
void GrUsrCustomCircleArc(int xc,int yc,int r,int start,int end,
                          int style,const GrLineOption *o);
void GrUsrCustomEllipseArc(int xc,int yc,int xa,int ya,int start,int end,
                           int style,const GrLineOption *o);
void GrUsrCustomPolyLine(int numpts,int points[][2],const GrLineOption *o);
void GrUsrCustomPolygon(int numpts,int points[][2],const GrLineOption *o);
void GrUsrPatternedLine(int x1,int y1,int x2,int y2,GrLinePattern *lp);
void GrUsrPatternedBox(int x1,int y1,int x2,int y2,GrLinePattern *lp);
void GrUsrPatternedCircle(int xc,int yc,int r,GrLinePattern *lp);
void GrUsrPatternedEllipse(int xc,int yc,int xa,int ya,GrLinePattern *lp);
void GrUsrPatternedCircleArc(int xc,int yc,int r,int start,int end,
                             int style,GrLinePattern *lp);
void GrUsrPatternedEllipseArc(int xc,int yc,int xa,int ya,int start,int end,
                              int style,GrLinePattern *lp);
void GrUsrPatternedPolyLine(int numpts,int points[][2],GrLinePattern *lp);
void GrUsrPatternedPolygon(int numpts,int points[][2],GrLinePattern *lp);
void GrUsrPatternFilledPlot(int x,int y,GrPattern *p);
void GrUsrPatternFilledLine(int x1,int y1,int x2,int y2,GrPattern *p);
void GrUsrPatternFilledBox(int x1,int y1,int x2,int y2,GrPattern *p);
void GrUsrPatternFilledCircle(int xc,int yc,int r,GrPattern *p);
void GrUsrPatternFilledEllipse(int xc,int yc,int xa,int ya,GrPattern *p);
void GrUsrPatternFilledCircleArc(int xc,int yc,int r,int start,int end,int style,GrPat
void GrUsrPatternFilledEllipseArc(int xc,int yc,int xa,int ya,int start,int end,int st
void GrUsrPatternFilledConvexPolygon(int numpts,int points[][2],GrPattern *p);
void GrUsrPatternFilledPolygon(int numpts,int points[][2],GrPattern *p);
void GrUsrPatternFloodFill(int x, int y, GrColor border, GrPattern *p);
void GrUsrDrawChar(int chr,int x,int y,const GrTextOption *opt);
void GrUsrDrawString(char *text,int length,int x,int y,const GrTextOption *opt);
void GrUsrTextXY(int x,int y,char *text,GrColor fg,GrColor bg);
```

Graphics cursors

The library provides support for the creation and usage of an unlimited number of graphics cursors. An application can use these cursors for any purpose. Cursors always save the area they occupy before they are drawn. When moved or erased they restore this area. As a general rule of thumb, an application should erase a cursor before making changes to an area it occupies and redraw the cursor after finishing the drawing. Cursors are created with the GrBuildCursor function:

```
GrCursor *GrBuildCursor(char far *pixels,int pitch,int w,int h,
                        int xo,int yo,const GrColorTableP c);
```

The pixels, w (=width), h (=height) and c (= color table) arguments are similar to the arguments of the pixmap building library function GrBuildPixmap (see that paragraph for a more detailed explanation.), but with two differences. First, is not assumed that the pixels data is w x h sized, the pitch argument set the offset between rows. Second, the pixmap data is interpreted slightly differently, any pixel with value zero is taken as a "transparent" pixel, i.e. the background will show through the cursor pattern at that pixel. A pixmap data byte with value = 1 will refer to the first color in the table, and so on.

The xo (= X offset) and yo (= Y offset) arguments specify the position (from the top left corner of the cursor pattern) of the cursor's "hot point".

The GrCursor data structure:

```
typedef struct _GR_cursor {
  struct _GR_context work;            /* work areas (4) */
  int     xcord,ycord;                /* cursor position on screen */
  int     xsize,ysize;                /* cursor size */
  int     xoffs,yoffs;                /* LU corner to hot point offset */
  int     xwork,ywork;                /* save/work area sizes */
  int     xwpos,ywpos;                /* save/work area position on screen */
  int     displayed;                  /* set if displayed */
} GrCursor;
```

is typically not used (i.e. read or changed) by the application program, it should just pass pointers to these structures to the appropriate library functions. Other cursor manipulation functions:

```
void GrDisplayCursor(GrCursor *cursor);
void GrEraseCursor(GrCursor *cursor);
void GrMoveCursor(GrCursor *cursor,int x,int y);
void GrDestroyCursor(GrCursor *cursor);
```

See the see [curstest.c], page 51 example.

Keyboard input

GRX can handle platform independant key input. The file grxkeys.h defines the keys to be used in the user's program. This is an extract:

```
#define GrKey_Control_A            0x0001
#define GrKey_Control_B            0x0002
#define GrKey_Control_C            0x0003
...
#define GrKey_A                    0x0041
#define GrKey_B                    0x0042
#define GrKey_C                    0x0043
...
#define GrKey_F1                   0x013b
#define GrKey_F2                   0x013c
#define GrKey_F3                   0x013d
...
#define GrKey_Alt_F1               0x0168
#define GrKey_Alt_F2               0x0169
#define GrKey_Alt_F3               0x016a
```

But you can be confident that the standard ASCII is right maped. The GrKeyType type is defined to store keycodes:

```
typedef unsigned short GrKeyType;
```

This function:

```
int GrKeyPressed(void);
```

returns non zero if there are any keycode waiting, that can be read with:

```
GrKeyType GrKeyRead(void);
```

The function:

```
int GrKeyStat(void);
```

returns a keyboard status word, or-ing it with the next defines it can be known the status of some special keys:

```
#define GR_KB_RIGHTSHIFT    0x01      /* right shift key depressed */
#define GR_KB_LEFTSHIFT     0x02      /* left shift key depressed */
#define GR_KB_CTRL          0x04      /* CTRL depressed */
#define GR_KB_ALT           0x08      /* ALT depressed */
#define GR_KB_SCROLLOCK     0x10      /* SCROLL LOCK active */
#define GR_KB_NUMLOCK       0x20      /* NUM LOCK active */
#define GR_KB_CAPSLOCK      0x40      /* CAPS LOCK active */
#define GR_KB_INSERT        0x80      /* INSERT state active */
#define GR_KB_SHIFT         (GR_KB_LEFTSHIFT | GR_KB_RIGHTSHIFT)
```

See the see [keys.c], page 88 example.

Mouse event handling

All mouse services need the presence of a mouse. An application can test whether a mouse is available by calling the function:

```
int  GrMouseDetect(void);
```

which will return zero if no mouse (or mouse driver) is present, non-zero otherwise. The mouse must be initialized by calling one (and only one) of these functions:

```
void GrMouseInit(void);
void GrMouseInitN(int queue_size);
```

GrMouseInit sets a event queue (see below) size to GR_M_QUEU_SIZE (128). A user supply event queue size can be set calling GrMouseInitN instead.

It is a good practice to call GrMouseUnInit before exiting the program. This will restore any interrupt vectors hooked by the program to their original values.

```
void GrMouseUnInit(void);
```

The mouse can be controlled with the following functions:

```
void GrMouseSetSpeed(int spmult,int spdiv);
void GrMouseSetAccel(int thresh,int accel);
void GrMouseSetLimits(int x1,int y1,int x2,int y2);
void GrMouseGetLimits(int *x1,int *y1,int *x2,int *y2);
void GrMouseWarp(int x,int y);
```

The library calculates the mouse position only from the mouse mickey counters. (To avoid the limit and 'rounding to the next multiple of eight' problem with some mouse driver when it finds itself in a graphics mode unknown to it.) The parameters to the GrMouseSetSpeed function specify how coordinate changes are obtained from mickey counter changes, multipling by spmult and dividing by spdiv. In high resolution graphics modes the value of one just works fine, in low resolution modes (320x200 or similar) it is best set the spdiv to two or three. (Of course, it also depends on the sensitivity the mouse.) The GrMouseSetAccel function is used to control the ballistic effect: if a mouse coordinate changes between two samplings by more than the thresh parameter, the change is multiplied by the accel parameter. NOTE: some mouse drivers perform similar calculations before reporting the coordinates in mickeys. In this case the acceleration done by the library will be additional to the one already performed by the mouse driver. The limits of the mouse movement can be set (passed limits will be clipped to the screen) with GrMouseSetLimits (default is the whole screen) and the current limits can be obtained with GrMouseGetLimits. GrMouseWarp sets the mouse cursor to the specified position.

As typical mouse drivers do not know how to draw mouse cursors in high resolution graphics modes, the mouse cursor is drawn by the library. The mouse cursor can be set with:

```
void GrMouseSetCursor(GrCursor *cursor);
void GrMouseSetColors(GrColor fg,GrColor bg);
```

GrMouseSetColors uses an internal arrow pattern, the color fg will be used as the interior of it and bg will be the border. The current mouse cursor can be obtained with:

```
GrCursor *GrMouseGetCursor(void);
```

The mouse cursor can be displayed/erased with:

```
void GrMouseDisplayCursor(void);
void GrMouseEraseCursor(void);
```

The mouse cursor can be left permanently displayed. All graphics primitives except for the few non-clipping functions check for conflicts with the mouse cursor and erase it before the drawing if necessary. Of course, it may be more efficient to erase the cursor manually before a long drawing sequence and redraw it after completion. The library provides an alternative pair of calls for this purpose which will erase the cursor only if it interferes with the drawing:

```
int  GrMouseBlock(GrContext *c,int x1,int y1,int x2,int y2);
void GrMouseUnBlock(int return_value_from_GrMouseBlock);
```

GrMouseBlock should be passed the context in which the drawing will take place (the usual convention of NULL meaning the current context is supported) and the limits of the affected area. It will erase the cursor only if it interferes with the drawing. When the drawing is finished GrMouseUnBlock must be called with the argument returned by GrMouseBlock.

The status of the mouse cursor can be obtained with calling GrMouseCursorIsDisplayed. This function will return non-zero if the cursor is displayed, zero if it is erased.

```
int  GrMouseCursorIsDisplayed(void);
```

The library supports (beside the simple cursor drawing) three types of "rubberband" attached to the mouse cursor. The GrMouseSetCursorMode function is used to select the cursor drawing mode.

```
void GrMouseSetCursorMode(int mode,...);
```

The parameter mode can have the following values:

```
#define GR_M_CUR_NORMAL   0    /* MOUSE CURSOR modes: just the cursor */
#define GR_M_CUR_RUBBER   1    /* rect. rubber band (XOR-d to the screen) */
#define GR_M_CUR_LINE     2    /* line attached to the cursor */
#define GR_M_CUR_BOX      3    /* rectangular box dragged by the cursor */
```

GrMouseSetCursorMode takes different parameters depending on the cursor drawing mode selected. The accepted call formats are:

```
GrMouseSetCursorMode(M_CUR_NORMAL);
GrMouseSetCursorMode(M_CUR_RUBBER,xanchor,yanchor,GrColor);
GrMouseSetCursorMode(M_CUR_LINE,xanchor,yanchor,GrColor);
GrMouseSetCursorMode(M_CUR_BOX,dx1,dy1,dx2,dy2,GrColor);
```

The anchor parameters for the rubberband and rubberline modes specify a fixed screen location to which the other corner of the primitive is bound. The dx1 through dy2 parameters define the offsets of the corners of the dragged box from the hotpoint of the mouse cursor. The color value passed is always XOR-ed to the screen, i.e. if an application wants the rubberband to appear in a given color on a given background then it has to pass the XOR of these two colors to GrMouseSetCursorMode.

The GrMouseGetEvent function is used to obtain the next mouse or keyboard event. It takes a flag with various bits encoding the type of event needed. It returns the event in a GrMouseEvent structure. The relevant declarations from grx20.h:

```
void GrMouseGetEvent(int flags,GrMouseEvent *event);

typedef struct _GR_mouseEvent {    /* mouse event buffer structure */
  int  flags;                       /* event type flags (see above) */
  int  x,y;                         /* mouse coordinates */
  int  buttons;                     /* mouse button state */
  int  key;                         /* key code from keyboard */
  int  kbstat;                      /* keybd status (ALT, CTRL, etc..) */
  long dtime;                       /* time since last event (msec) */
} GrMouseEvent;
```

The event structure has been extended with a keyboard status word (thus a program can check for combinations like ALT-<left mousebutton press>) and a time stamp which can be used to check for double clicks, etc... The following macros have been defined in grx20.h to help in creating the control flag for GrMouseGetEvent and decoding the various bits in the event structure:

```
#define GR_M_MOTION         0x001         /* mouse event flag bits */
#define GR_M_LEFT_DOWN      0x002
#define GR_M_LEFT_UP        0x004
#define GR_M_RIGHT_DOWN     0x008
#define GR_M_RIGHT_UP       0x010
#define GR_M_MIDDLE_DOWN    0x020
#define GR_M_MIDDLE_UP      0x040
#define GR_M_BUTTON_DOWN    (GR_M_LEFT_DOWN | GR_M_MIDDLE_DOWN | \
                             GR_M_RIGHT_DOWN)
#define GR_M_BUTTON_UP      (GR_M_LEFT_UP   | GR_M_MIDDLE_UP   | \
                             GR_M_RIGHT_UP)
#define GR_M_BUTTON_CHANGE  (GR_M_BUTTON_UP | GR_M_BUTTON_DOWN )

#define GR_M_LEFT           1             /* mouse button index bits */
#define GR_M_RIGHT          2
```

```
#define GR_M_MIDDLE         4

#define GR_M_KEYPRESS       0x080       /* other event flag bits */
#define GR_M_POLL           0x100
#define GR_M_NOPAINT        0x200
#define GR_M_EVENT          (GR_M_MOTION | GR_M_KEYPRESS | \
                             GR_M_BUTTON_DOWN | GR_M_BUTTON_UP)
```

GrMouseGetEvent will display the mouse cursor if it was previously erased and the GR_M_NOPAINT bit is not set in the flag passed to it. In this case it will also erase the cursor after an event has been obtained.

GrMouseGetEvent block until a event is produced, except if the GR_M_POLL bit is set in the falg passed to it.

Another version of GetEvent:

```
void GrMouseGetEventT(int flags,GrMouseEvent *event,long timout_msecs);
```

can be istructed to wait timout_msec for the presence of an event. Note that event->dtime is only valid if any event occured (event->flags != 0) otherwise it's set as -1. Additionally event timing is real world time even in X11 && Linux.

If there are one or more events waiting the function:

```
int  GrMousePendingEvent(void);
```

returns non-zero value.

The generation of mouse and keyboard events can be individually enabled or disabled (by passing a non-zero or zero, respectively, value in the corresponding enable_XX parameter) by calling:

```
void GrMouseEventEnable(int enable_kb,int enable_ms);
```

Note that GrMouseInit set both by default. If you want to use GrMouseGetEvent and GrKeyRead at the same time, a call to GrMouseEventEnable(0,1) is needed before input process.

See the see [mousetst.c], page 103 example.

Writing/reading PNM graphics files

GRX includes functions to load/save a context from/to a PNM file.

PNM is a group of simple graphics formats from the NetPbm (http://netpbm.sourceforge.net) distribution. NetPbm can convert from/to PNM lots of graphics formats, and apply some transformations to PNM files. (Note. You don't need the NetPbm distribution to use this functions).

There are six PNM formats:

```
P1 text PBM (bitmap)
P2 text PGM (gray scale)
P3 text PPM (real color)
P4 binary PBM (bitmap)
P5 binary PGM (gray scale)
P6 binary PPM (real color)
```

GRX can handle the binary formats only (get the NetPbm distribution if you need convert text to binary formats).

To save a context in a PNM file you have three functions:

```
int GrSaveContextToPbm( GrContext *grc, char *pbmfn, char *docn );
int GrSaveContextToPgm( GrContext *grc, char *pgmfn, char *docn );
int GrSaveContextToPpm( GrContext *grc, char *ppmfn, char *docn );
```

they work both in RGB and palette modes, grc must be a pointer to the context to be saved, if it is NULL the current context is saved; p-mfn is the file name to be created and docn is an optional text comment to be written in the file, it can be NULL. Three functions return 0 on succes and -1 on error.

GrSaveContextToPbm dumps a context in a PBM file (bitmap). If the pixel color isn't Black it asumes White.

GrSaveContextToPgm dumps a context in a PGM file (gray scale). The colors are quantized to gray scale using .299r + .587g + .114b.

GrSaveContextToPpm dumps a context in a PPM file (real color). To load a PNM file in a context you must use:

```
int GrLoadContextFromPnm( GrContext *grc, char *pnmfn );
```

it support reading PBM, PGM and PPM binary files. grc must be a pointer to the context to be written, if it is NULL the current context is used; p-mfn is the file name to be read. If context dimensions are lesser than pnm dimensions, the function loads as much as it can. If color mode is not in RGB mode, the routine allocates as much colors as it can. The function returns 0 on succes and -1 on error.

To query the file format, width and height of a PNM file you can use:

```
int GrQueryPnm( char *ppmfn, int *width, int *height, int *maxval );
```

pnmfn is the name of pnm file; width returns the pnm width; height returns the pnm height; maxval returns the max color component value. The function returns 1 to 6 on success (the PNM format) or -1 on error.

The two next functions:

```
int GrLoadContextFromPnmBuffer( GrContext *grc, const char *pnmbuf );
int GrQueryPnmBuffer( const char *pnmbuf, int *width, int *height, int *max-
val );
```

work like GrLoadContextFromPnm and GrQueryPnmBuffer, but they get his input from a buffer instead of a file. This way, pnm files can be embeded in a program (using the bin2c program by example).

Writing/reading PNG graphics files

GRX includes functions to load/save a context from/to a png file. But note, for this purpose it needs the (a href="http://www.libpng.org/pub/png/libpng.html) libpng library, and to enable the png support before make the GRX lib.

Use next function to save a context in a PNG file:

```
int GrSaveContextToPng( GrContext *grc, char *pngfn );
```

it works both in RGB and palette modes, grc must be a pointer to the context to be saved, if it is NULL the current context is saved; pngfn is the file name to be created. The function returns 0 on succes and -1 on error.

To load a PNG file in a context you must use:

```
int GrLoadContextFromPng( GrContext *grc, char *pngfn, int use_alpha );
```

grc must be a pointer to the context to be written, if it is NULL the current context is used; pngfn is the file name to be read; set use_alpha to 1 if you want to use the image alpha channel (if available). If context dimensions are lesser than png dimensions, the function loads as much as it can. If color mode is not in RGB mode, the routine allocates as much colors as it can. The function returns 0 on succes and -1 on error.

To query the width and height of a PNG file you can use:

```
int GrQueryPng( char *pngfn, int *width, int *height );
```

pngfn is the name of png file; width returns the png width; height returns the png height. The function returns 0 on success or -1 on error.

The function:

```
int GrPngSupport( void );
```

returns 1 if there is png support in the library, 0 otherwise. If there is not support for png, dummy functions are added to the library, returning error (-1) ever.

Writing/reading PNG graphics files

GRX includes functions to load/save a context from/to a jpeg file. But note, for this purpose it needs the (a href="http://www.ijg.org) libjpeg library, and to enable the jpeg support before make the GRX lib.

Use next function to save a context in a JPEG file:

```
int GrSaveContextToJpeg( GrContext *grc, char *jpegfn, int quality );
```

it works both in RGB and palette modes, grc must be a pointer to the context to be saved, if it is NULL the current context is saved; jpegfn is the file name to be created; quality is a number between 1 and 100 to drive the compression quality, use higher values for better quality (and bigger files), you can use 75 as a standard value, normally a value between 50 and 95 is good. The function returns 0 on succes and -1 on error.

This function saves a context in a grayscale JPEG file:

```
int GrSaveContextToGrayJpeg( GrContext *grc, char *jpegfn, int quality );
```

parameters and return codes are like in GrSaveContextToJpeg. The colors are quantized to gray scale using .299r + .587g + .114b.

To load a JPEG file in a context you must use:

```
int GrLoadContextFromJpeg( GrContext *grc, char *jpegfn, int scale );
```

grc must be a pointer to the context to be written, if it is NULL the current context is used; jpegfn is the file name to be read; set scale to 1, 2, 4 or 8 to reduce the loaded image to 1/1, 1/2, 1/4 or 1/8. If context dimensions are lesser than jpeg dimensions, the function loads as much as it can. If color mode is not in RGB mode, the routine allocates as much colors as it can. The function returns 0 on succes and -1 on error.

To query the width and height of a JPEG file you can use:

```
int GrQueryJpeg( char *jpegfn, int *width, int *height );
```

jpegfn is the name of jpeg file; width returns the jpeg width; height returns the jpeg height. The function returns 0 on success or -1 on error.

The function:

```
int GrJpegSupport( void );
```

returns 1 if there is jpeg support in the library, 0 otherwise. If there is not support for jpeg, dummy functions are added to the library, returning error (-1) ever.

Miscellaneous functions

Here we will describe some miscellaneous functions.

```
unsigned GrGetLibraryVersion(void);
```

GrGetLibraryVersion returns the GRX version API, like a hexadecimal coded number. By example 0x0241 means 2.4.1 Because grx20.h defines the GRX_VERSION_API macro, you can check if both, the library and the include file, are in the same version using if(GrGetLibraryVersion() == GRX_VERSION_API)

```
unsigned GrGetLibrarySystem(void);
```

This functions returns a unsigned integer identifing the system you are working in. grx20.h defines some macros you can use:

```
/* these are the supported configurations: */
#define GRX_VERSION_TCC_8086_DOS        1   /* also works with BCC */
#define GRX_VERSION_GCC_386_DJGPP       2   /* DJGPP v2 */
#define GRX_VERSION_GCC_386_LINUX       3   /* the real stuff */
#define GRX_VERSION_GENERIC_X11         4   /* generic X11 version */
#define GRX_VERSION_WATCOM_DOS4GW       5   /* GS - Watcom C++ 11.0 32 Bit
#define GRX_VERSION_GCC_386_WIN32       7   /* WIN32 using Mingw32 */
#define GRX_VERSION_MSC_386_WIN32       8   /* WIN32 using MS-VC */
#define GRX_VERSION_GCC_386_CYG32       9   /* WIN32 using CYGWIN */
#define GRX_VERSION_GCC_386_X11        10   /* X11 version */
#define GRX_VERSION_GCC_X86_64_LINUX   11   /* console framebuffer 64 */
#define GRX_VERSION_GCC_X86_64_X11     12   /* X11 version 64 */
```

Note. On Linux, GrGetLibrarySystem returns GRX_VERSION_GCC_386_LINUX even in the X11 version.

```
void GrSetWindowTitle(char *title);
```

GrSetWindowTitle sets the main window title in the X11 an Win32 versions. It doesn't do nothing in the DOS and Linux-SvgaLib versions.

```
void GrSleep(int msec);
```

This function stops the program execution for msec miliseconds.

```
 GrContext *GrCreateFrameContext(GrFrameMode md,int w,int h,
            char far *memory[4],GrContext *where);
```

This function is like GrCreateContext, except that you can specify any valid memory frame mode, not only the Screen associated frame mode. It can be used for special purposes (see GrBitBlt1bpp for an example).

```
void GrBitBlt1bpp(GrContext *dst,int dx,int dy,GrContext *src,
     int x1,int y1,int x2,int y2,GrColor fg,GrColor bg);
```

This special function does a bitblt from a 1bpp context (a bitmap really), using fg and bg like the color+opcode when bit=1 and bit=0 respectively. Here is an example:

```
pContext = GrCreateFrameContext(GR_frameRAM1, sizex, sizey, NULL, NULL);
/* draw something (black and white) into the bitmap */
GrSetContext(pContext);
GrClearContext( GrBlack() );
GrLine(0, 0, sizex-1, sizey-1, GrWhite());
GrLine(0, sizey-1, sizex-1, 0, GrWhite());

/* Put the bitmap into the screen */
GrSetContext(NULL);
fcolor = GrAllocColor( 255,0,0 );
bcolor = GrAllocColor( 0,0,255 );
GrBitBlt1bpp(NULL,x,y,pContext,0,0,sizex-1,sizey-1,fcolor,bcolor);
```

BGI interface

From the 2.3.1 version, GRX includes the BCC2GRX library created by Hartmut Schirmer. The BCC2GRX was created to allow users of GRX to compile graphics programs written for Borland-C++ and Turbo-C graphics interface. BCC2GRX is not a convenient platform to develop new BGI programs. Of course you should use native GRX interface in such cases!

Read the readme.bgi file for more info.

Test examples

arctest.c

```
/**
 ** arctest.c ---- test arc outline and filled arc drawing
 **
 ** Copyright (c) 1995 Csaba Biegl, 820 Stirrup Dr, Nashville, TN 37221
 ** [e-mail: csaba@vuse.vanderbilt.edu]
 **
 ** This is a test/demo file of the GRX graphics library.
 ** You can use GRX test/demo files as you want.
 **
 ** The GRX graphics library is free software; you can redistribute it
 ** and/or modify it under some conditions; see the "copying.grx" file
 ** for details.
 **
 ** This library is distributed in the hope that it will be useful,
 ** but WITHOUT ANY WARRANTY; without even the implied warranty of
 ** MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 **
 **/

#include <string.h>
#include "test.h"  /* see [test.h], page 150 */

TESTFUNC(arctest)
{
char buff[300];
int  xc,yc,xa,ya,start,end;
FILE *fp;
GrColor red   = GrAllocColor(255,0,0);
GrColor green = GrAllocColor(0,255,0);
GrColor blue  = GrAllocColor(0,0,255);

fp = fopen("arctest.dat","r");  /* see [arctest.dat], page 152 */
if(fp == NULL) return;
while(fgets(buff,299,fp) != NULL) {
    int len = strlen(buff);
    while(--len >= 0) {
if(buff[len] == '\n') { buff[len] = '\0'; continue; }
if(buff[len] == '\r') { buff[len] = '\0'; continue; }
break;
    }
    if(sscanf(buff,
      "arc xc=%d yc=%d xa=%d ya=%d start=%d end=%d",
      &xc,&yc,&xa,&ya,&start,&end) == 6) {
GrClearScreen(GrBlack());
GrEllipse(xc,yc,xa,ya,red);
GrFilledEllipse(xc,yc,xa,ya,blue);
GrEllipseArc(xc,yc,xa,ya,start,end,GR_ARC_STYLE_CLOSE2,GrWhite());
GrTextXY(0,0,buff,GrWhite(),GrNOCOLOR);
                GrTextXY(0,20,"press any key to continue",GrWhite(),GrNOCOLOR);
GrKeyRead();
GrClearScreen(GrBlack());
GrEllipseArc(xc,yc,xa,ya,start,end,GR_ARC_STYLE_CLOSE2,red);
GrFilledEllipseArc(xc,yc,xa,ya,start,end,GR_ARC_STYLE_CLOSE2,green);
GrTextXY(0,0,buff,GrWhite(),GrNOCOLOR);
                GrTextXY(0,20,"press any key to continue",GrWhite(),GrNOCOLOR);
GrKeyRead();
    }
}
fclose(fp);
}
```

bb1test.c

```
/**
 ** bb1test.c ---- test the GrBitBlt1bpp routine
 **
 ** Copyright (c) 2001 Josu Onandia
 ** [e-mail: jonandia@fagorautomation.es].
 **
 ** This is a test/demo file of the GRX graphics library.
 ** You can use GRX test/demo files as you want.
 **
 ** The GRX graphics library is free software; you can redistribute it
 ** and/or modify it under some conditions; see the "copying.grx" file
 ** for details.
 **
 ** This library is distributed in the hope that it will be useful,
 ** but WITHOUT ANY WARRANTY; without even the implied warranty of
 ** MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 **/

#include <stdlib.h>
#include "grx20.h"
#include "grxkeys.h"

int main(void)
{
  GrContext *pContext;
  int sizex = 40;
  int sizey = 40;
  int x = 0;
  int y = 40;
  GrColor fcolor, bcolor;
  GrKeyType k;

  GrSetMode( GR_default_graphics );
  /* Create a 1bpp bitmap */
  pContext = GrCreateFrameContext(GR_frameRAM1, sizex, sizey, NULL, NULL);
  /* draw something (black and white) into the bitmap */
  GrSetContext(pContext);
  GrClearContext( GrBlack() );
  GrLine(0, 0, sizex-1, sizey-1, GrWhite());
  GrLine(0, sizey-1, sizex-1, 0, GrWhite());

  GrSetContext(NULL);
  fcolor = GrAllocColor( 255,0,0 );
  bcolor = GrAllocColor( 0,0,255 );
  GrTextXY(0,0,"Type u d l r U D L R to move, 1 2 to change color, q to quit",
            GrWhite(),GrNOCOLOR);
  GrSetClipBox(0, 40, GrScreenX(), GrScreenY());

  /* Put the bitmap into the screen */
  GrBitBlt1bpp(NULL,x,y,pContext,0,0,sizex-1,sizey-1,fcolor,bcolor);

  while( 1 ){
    k = GrKeyRead();
    if( k == ’q’ ) break;
    switch( k ) {
      case ’u’: y--; break;
      case ’d’: y++; break;
      case ’l’: x--; break;
      case ’r’: x++; break;
      case ’U’: y -= 10; break;
      case ’D’: y += 10; break;
      case ’L’: x -= 10; break;
      case ’R’: x += 10; break;
      case ’1’: fcolor = GrAllocColor( 255,0,0 );
                bcolor = GrAllocColor( 0,0,255 );
                break;
      case ’2’: fcolor = GrAllocColor( 0,255,255 );
                bcolor = GrAllocColor( 255,255,0 );
                break;
      default:  continue;
      }
    if(x < -40) x = -40;
    if(x > GrScreenX()) x = GrScreenX();
    if(y < 0) y = 0;
    if(y > GrScreenY()) y = GrScreenY();
    GrBitBlt1bpp(NULL,x,y,pContext,0,0,sizex-1,sizey-1,fcolor,bcolor);
    }

  /* Destroy */
  GrDestroyContext(pContext);

  GrSetMode(GR_default_text);
  return 0;
}
```

blittest.c

```
/**
 ** blittest.c ---- test various bitblt-s
 **
 ** Copyright (c) 1995 Csaba Biegl, 820 Stirrup Dr, Nashville, TN 37221
 ** [e-mail: csaba@vuse.vanderbilt.edu]
 **
 ** This is a test/demo file of the GRX graphics library.
 ** You can use GRX test/demo files as you want.
 **
 ** The GRX graphics library is free software; you can redistribute it
 ** and/or modify it under some conditions; see the "copying.grx" file
 ** for details.
 **
 ** This library is distributed in the hope that it will be useful,
 ** but WITHOUT ANY WARRANTY; without even the implied warranty of
 ** MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 **
 **/

#include <stdlib.h>
#include <string.h>

#include "test.h"  /* see [test.h], page 150 */

#define BHH     (GrScreenY() / 10)
int     BWW =   83;

void drbox(GrContext *src,int x,int y)
{
GrColor c1 = GrAllocColor(0,0,255);
GrColor c2 = GrAllocColor(255,0,0);
int  xx;

GrClearScreen(c1);
GrSetContext(src);
GrSetClipBox(x-10,y-10,x+BWW-1+10,y+BHH-1+10);
GrClearClipBox(c2);
GrSetClipBox(x,y,x+BWW-1,y+BHH-1);
GrClearClipBox(GrBlack());
GrBox(x,y,x+BWW-1,y+BHH-1,GrWhite());
for(xx = x; xx < x+BWW; xx += 5) {
    GrLine(xx,y,xx+BHH,y+BHH,GrWhite());
    GrLine(xx,y,xx-BHH,y+BHH,GrWhite());
}
GrSetContext(NULL);
GrResetClipBox();
}

void doblits(GrContext *src,int x,int y)
{
int xx = (GrSizeX() - BWW)/ 2;
int yy = 2;
int ii;

for(ii = 0; ii < 8; ii++) {
    GrBitBlt(NULL,xx,yy,src,x,y,x+BWW-1,y+BHH-1,GrWRITE);
    xx++;
    yy += (BHH + 2);
}
/*
  {
GrColor xc = GrAllocColor(255,255,255) | GrXOR;
        GrKeyRead();
xx = (GrSizeX() - BWW)/ 2;
yy = 2;
for(ii = 0; ii < 8; ii++) {
    GrFilledBox(xx,yy,xx+BWW-1,yy+BHH-1,xc);
    xx++;
    yy += (BHH + 2);
}
  }
*/
}

void bltest(GrContext *src,int x,int y)
{
int ii;

for(ii = 0; ii < 8; ii++) {
    drbox(src,x,y);
            doblits(src,x,y);
            GrKeyRead();
    x++;
}
}

void blxtest(void)
{
GrContext memc;
int cw = (BWW + 28) & ~7;
int ch = BHH + 20;

bltest(NULL,GrScreenX()-BWW-8,GrScreenY()-BHH);
bltest(NULL,0,0);
GrCreateContext(cw,ch,NULL,&memc);
bltest(&memc,cw-BWW-8,ch-BHH);
}

TESTFUNC(blittest)
```

```
{
GrFBoxColors bcolors,ocolors,icolors;
GrColor c,bg;
int  x = GrSizeX();
int  y = GrSizeY();
int  ww = (x * 2) / 3;
int  wh = (y * 2) / 3;
int  ii,jj;
int  wdt = ww / 150;
int  bw = x / 17;
int  bh = y / 17;
int  bx,by;
int  cnt;

GrContext *save = GrCreateSubContext(0,0,GrMaxX(),GrMaxY(),NULL,NULL);
GrContext *tile = GrCreateContext(bw,bh,NULL,NULL);

blxtest();
        GrKeyRead();

BWW = 3;
blxtest();
        GrKeyRead();

bcolors.fbx_intcolor = GrAllocColor(160,100,30);
bcolors.fbx_topcolor = GrAllocColor(240,150,45);
bcolors.fbx_leftcolor = GrAllocColor(240,150,45);
bcolors.fbx_rightcolor = GrAllocColor(80,50,15);
bcolors.fbx_bottomcolor = GrAllocColor(80,50,15);

ocolors.fbx_intcolor = GrAllocColor(0,120,100);
ocolors.fbx_topcolor = GrAllocColor(0,180,150);
ocolors.fbx_leftcolor = GrAllocColor(0,180,150);
ocolors.fbx_rightcolor = GrAllocColor(0,90,60);
ocolors.fbx_bottomcolor = GrAllocColor(0,90,60);

icolors.fbx_intcolor = bg = GrAllocColor(30,30,30);
icolors.fbx_bottomcolor = GrAllocColor(0,180,150);
icolors.fbx_rightcolor = GrAllocColor(0,180,150);
icolors.fbx_leftcolor = GrAllocColor(0,90,60);
icolors.fbx_topcolor = GrAllocColor(0,90,60);

c = GrAllocColor(250,250,0);

for(ii = 0,by = -(bh/3); ii < 19; ii++) {
    for(jj = 0,bx = -(bw/2); jj < 19; jj++) {
GrFramedBox(bx+2*wdt,by+2*wdt,bx+bw-2*wdt-1,by+bh-2*wdt-1,2*wdt,&bcolors);
bx += bw;
    }
    by += bh;
}

GrFramedBox(ww/4-5*wdt-1,wh/4-5*wdt-1,ww/4+5*wdt+ww+1,wh/4+5*wdt+wh+1,wdt,&ocolors);
GrFramedBox(ww/4-1,wh/4-1,ww/4+ww+1,wh/4+wh+1,wdt,&icolors);

GrSetClipBox(ww/4,wh/4,ww/4+ww,wh/4+wh);
drawing(ww/4,wh/4,ww,wh,c,bg);
        GrKeyRead();

GrClearScreen(0);
GrSetContext(save);

bx = -(bw/2) + 15*bw;
by = -(bh/3) + 15*bh;

GrFramedBox(bx+2*wdt,by+2*wdt,bx+bw-2*wdt-1,by+bh-2*wdt-1,2*wdt,&bcolors);

for (cnt=0; cnt<3; cnt++) {
  for(ii = 0,by = -(bh/3); ii < 19; ii++) {
    for(jj = 0,bx = -(bw/2); jj < 19; jj++) {
if((ii != 15) || (jj != 15)) {
    GrBitBlt(save,
bx,by,
save,
-(bw/2) + 15*bw,
-(bh/3) + 15*bh,
-(bw/2) + 15*bw + bw - 1,
-(bh/3) + 15*bh + bh - 1,
cnt==1 ? GrXOR : GrWRITE
    );
}
bx += bw;
    }
    by += bh;
  }
}

GrFramedBox(ww/4-5*wdt-1,wh/4-5*wdt-1,ww/4+5*wdt+ww+1,wh/4+5*wdt+wh+1,wdt,&ocolors);
GrFramedBox(ww/4-1,wh/4-1,ww/4+ww+1,wh/4+wh+1,wdt,&icolors);

GrSetClipBox(ww/4,wh/4,ww/4+ww,wh/4+wh);
drawing(ww/4,wh/4,ww,wh,c,bg);
        GrKeyRead();
```

```
GrBitBlt(tile,
    0,0,
    save,
    -(bw/2) + 15*bw,
    -(bh/3) + 15*bh,
    -(bw/2) + 15*bw + bw - 1,
    -(bh/3) + 15*bh + bh - 1,
    GrWRITE
);
GrSetContext(tile);
GrFramedBox(2*wdt,2*wdt,bw-2*wdt-1,bh-2*wdt-1,2*wdt,&bcolors);

GrClearScreen(0);
GrSetContext(save);

for(ii = 0,by = -(bh/3); ii < 19; ii++) {
    for(jj = 0,bx = -(bw/2); jj < 19; jj++) {
GrBitBlt(save,
    bx,by,
    tile,
    0,0,
    bw-1,bh-1,
    GrWRITE
);
bx += bw;
    }
    by += bh;
}

GrFramedBox(ww/4-5*wdt-1,wh/4-5*wdt-1,ww/4+5*wdt+ww+1,wh/4+5*wdt+wh+1,wdt,&ocolors);
GrFramedBox(ww/4-1,wh/4-1,ww/4+ww+1,wh/4+wh+1,wdt,&icolors);

GrSetClipBox(ww/4,wh/4,ww/4+ww,wh/4+wh);
drawing(ww/4,wh/4,ww,wh,c,bg);

        GrKeyRead();
GrResetClipBox();
GrBitBlt(NULL,
   60,60,
   NULL,
   20,20,
   GrSizeX() - 40,
   GrSizeY() - 40,
   GrWRITE
);
```

```
        GrKeyRead();

GrBitBlt(NULL,
   10,10,
   NULL,
   60,60,
   GrSizeX() - 40,
   GrSizeY() - 40,
   GrWRITE
);

        GrKeyRead();

GrSetContext(tile);
GrClearContext(0);

GrBitBlt(tile,
    0,0,
    save,
    -(bw/2),
    -(bh/3),
    -(bw/2) + 15*bw + bw - 1,
    -(bh/3) + 15*bh + bh - 1,
    GrWRITE
);

GrSetContext(save);
GrClearScreen(0);

for(ii = 0,by = -(bh/3); ii < 18; ii++) {
    for(jj = 0,bx = -(bw/2); jj < 18; jj++) {
GrBitBlt(save,
    bx,by,
    tile,
    0,0,
    bw-1,bh-1,
    GrWRITE
);
bx += bw;
    }
    by += bh;
}

GrFramedBox(ww/4-5*wdt-1,wh/4-5*wdt-1,ww/4+5*wdt+ww+1,wh/4+5*wdt+wh+1,wdt,&ocolors);
GrFramedBox(ww/4-1,wh/4-1,ww/4+ww+1,wh/4+wh+1,wdt,&icolors);

GrSetClipBox(ww/4,wh/4,ww/4+ww,wh/4+wh);
```

```
drawing(ww/4,wh/4,ww,wh,c,bg);

        GrKeyRead();

}
```

circtest.c

```
/**
 ** circtest.c ---- test circle and ellipse rendering
 **
 ** Copyright (c) 1995 Csaba Biegl, 820 Stirrup Dr, Nashville, TN 37221
 ** [e-mail: csaba@vuse.vanderbilt.edu]
 **
 ** This is a test/demo file of the GRX graphics library.
 ** You can use GRX test/demo files as you want.
 **
 ** The GRX graphics library is free software; you can redistribute it
 ** and/or modify it under some conditions; see the "copying.grx" file
 ** for details.
 **
 ** This library is distributed in the hope that it will be useful,
 ** but WITHOUT ANY WARRANTY; without even the implied warranty of
 ** MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 **
 **/

#include "test.h"  /* see [test.h], page 150 */
#include <math.h>

void drawellip(int xc,int yc,int xa,int ya,GrColor c1,GrColor c2,GrColor c3)
{
double ddx = (double)xa;
double ddy = (double)ya;
double R2 = ddx*ddx*ddy*ddy;
double SQ;
int x1,x2,y1,y2;
int dx,dy;

GrFilledBox(xc-xa,yc-ya,xc+xa,yc+ya,c1);
dx = xa;
dy = 0;
GrPlot(xc-dx,yc,c3);
GrPlot(xc+dx,yc,c3);
while(++dy <= ya) {
    SQ = R2 - (double)dy * (double)dy * ddx * ddx;
    dx = (int)(sqrt(SQ)/ddy + 0.5);
    x1 = xc - dx;
    x2 = xc + dx;
    y1 = yc - dy;
    y2 = yc + dy;
    GrPlot(x1,y1,c3);
    GrPlot(x2,y1,c3);
    GrPlot(x1,y2,c3);
    GrPlot(x2,y2,c3);
}
GrEllipse(xc,yc,xa,ya,c2);
}

TESTFUNC(circtest)
{
int     xc,yc;
int     xr,yr;
GrColor c1,c2,c3;

c1 = GrAllocColor(64,64,255);
c2 = GrAllocColor(255,255,64);
c3 = GrAllocColor(255,64,64);
xc = GrSizeX() / 2;
yc = GrSizeY() / 2;
xr = 1;
yr = 1;
while((xr < 1000) || (yr < 1000)) {
    drawellip(xc,yc,xr,yr,c1,c2,c3);
    xr += xr/4+1;
    yr += yr/4+1;
            GrSleep(200);
}
c1 = GrAllocColor(64,64,128);
xr = 4;
yr = 1;
while((xr < 1000) || (yr < 1000)) {
    drawellip(xc,yc,xr,yr,c1,c2,c3);
    yr += yr/4+1;
    xr = yr * 4;
            GrSleep(200);
}
c1 = GrAllocColor(64,64,64);
xr = 1;
yr = 4;
while((xr < 1000) || (yr < 1000)) {
    drawellip(xc,yc,xr,yr,c1,c2,c3);
    xr += xr/4+1;
    yr = xr * 4;
            GrSleep(200);
}

        GrTextXY(0,0,"press any key to continue",GrWhite(),GrBlack());
        GrKeyRead();
}
```

cliptest.c

```
/**
 ** cliptest.c ---- test clipping
 **
 ** Copyright (c) 1995 Csaba Biegl, 820 Stirrup Dr, Nashville, TN 37221
 ** [e-mail: csaba@vuse.vanderbilt.edu]
 **
 ** This is a test/demo file of the GRX graphics library.
 ** You can use GRX test/demo files as you want.
 **
 ** The GRX graphics library is free software; you can redistribute it
 ** and/or modify it under some conditions; see the "copying.grx" file
 ** for details.
 **
 ** This library is distributed in the hope that it will be useful,
 ** but WITHOUT ANY WARRANTY; without even the implied warranty of
 ** MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 **
 **/

#include "test.h"  /* see [test.h], page 150 */
#include "rand.h"  /* see [rand.h], page 149 */

TESTFUNC(cliptest)
{
long delay;
int x = GrSizeX();
int y = GrSizeY();
int ww = (x * 2) / 3;
int wh = (y * 2) / 3;
GrColor c;

c = GrAllocColor(200,100,100);
GrBox(ww/4-1,wh/4-1,ww/4+ww+1,wh/4+wh+1,GrWhite());
GrSetClipBox(ww/4,wh/4,ww/4+ww,wh/4+wh);

drawing(0,0,ww,wh,c,GrBlack());
GrKeyRead();

while(!GrKeyPressed()) {
    GrFilledBox(0,0,x,y,GrBlack());
    drawing(-(RND()%(2*ww))+ww/2,
-(RND()%(2*wh))+wh/2,
RND()%(3*ww)+10,
RND()%(3*wh)+10,
c,
GrNOCOLOR
    );
    for(delay = 200000L; delay > 0L; delay--);
}
GrKeyRead();
}
```

colorops.c

```
/**
 ** colorops.c ---- test WRITE, XOR, OR, and AND draw modes
 **
 ** Copyright (c) 1995 Csaba Biegl, 820 Stirrup Dr, Nashville, TN 37221
 ** [e-mail: csaba@vuse.vanderbilt.edu]
 **
 ** This is a test/demo file of the GRX graphics library.
 ** You can use GRX test/demo files as you want.
 **
 ** The GRX graphics library is free software; you can redistribute it
 ** and/or modify it under some conditions; see the "copying.grx" file
 ** for details.
 **
 ** This library is distributed in the hope that it will be useful,
 ** but WITHOUT ANY WARRANTY; without even the implied warranty of
 ** MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 **
 **/

#include "test.h"  /* see [test.h], page 150 */
#include "rand.h"  /* see [rand.h], page 149 */

TESTFUNC(colorops)
{
GrFBoxColors bcolors,ocolors,icolors;
GrColor bg,c;
int x = GrSizeX();
int y = GrSizeY();
int ww = (x * 2) / 3;
int wh = (y * 2) / 3;
int ii,jj;
int wdt = ww / 150;
int bw = x / 16;
int bh = y / 16;
int bx,by;

/* This won't work very well under X11 in pseudocolor
** mode (256 colors or less) if not using a private
** color map. The missing colors break RGB mode      */
GrSetRGBcolorMode();

bcolors.fbx_intcolor = GrAllocColor(160,100,30);
bcolors.fbx_topcolor = GrAllocColor(240,150,45);
bcolors.fbx_leftcolor = GrAllocColor(240,150,45);
bcolors.fbx_rightcolor = GrAllocColor(80,50,15);
bcolors.fbx_bottomcolor = GrAllocColor(80,50,15);

ocolors.fbx_intcolor = GrAllocColor(0,120,100);
ocolors.fbx_topcolor = GrAllocColor(0,180,150);
ocolors.fbx_leftcolor = GrAllocColor(0,180,150);
ocolors.fbx_rightcolor = GrAllocColor(0,90,60);
ocolors.fbx_bottomcolor = GrAllocColor(0,90,60);

icolors.fbx_intcolor = GrAllocColor(30,30,30);
icolors.fbx_bottomcolor = GrAllocColor(0,180,150);
icolors.fbx_rightcolor = GrAllocColor(0,180,150);
icolors.fbx_leftcolor = GrAllocColor(0,90,60);
icolors.fbx_topcolor = GrAllocColor(0,90,60);

c  = GrAllocColor(250,250,0);
bg = GrNOCOLOR;

for(ii = 0,by = -(bh / 3); ii < 17; ii++) {
    for(jj = 0,bx = (-bw / 2); jj < 17; jj++) {
GrFramedBox(bx+2*wdt,by+2*wdt,bx+bw-2*wdt-1,by+bh-2*wdt-1,2*wdt,&bcolors);█
bx += bw;
    }
    by += bh;
}

GrFramedBox(ww/4-5*wdt-1,wh/4-5*wdt-1,ww/4+5*wdt+ww+1,wh/4+5*wdt+wh+1,wdt,&ocolors);█
GrFramedBox(ww/4-1,wh/4-1,ww/4+ww+1,wh/4+wh+1,wdt,&icolors);

GrSetClipBox(ww/4,wh/4,ww/4+ww,wh/4+wh);
```

```
drawing(ww/4,wh/4,ww,wh,c,bg);
while(!GrKeyPressed()) {
    drawing(ww/4+(RND()%100),
wh/4+(RND()%100),
ww,
wh,
((RND() / 16) & (GrNumColors() - 1)),
bg
    );
}
GrKeyRead();
GrFramedBox(ww/4-1,wh/4-1,ww/4+ww+1,wh/4+wh+1,wdt,&icolors);
drawing(ww/4,wh/4,ww,wh,c,bg);
while(!GrKeyPressed()) {
    drawing(ww/4+(RND()%100),
wh/4+(RND()%100),
ww,
wh,
((RND() / 16) & (GrNumColors() - 1)) | GrXOR,
bg
    );
}
GrKeyRead();
GrFramedBox(ww/4-1,wh/4-1,ww/4+ww+1,wh/4+wh+1,wdt,&icolors);
drawing(ww/4,wh/4,ww,wh,c,bg);
while(!GrKeyPressed()) {
    drawing(ww/4+(RND()%100),
wh/4+(RND()%100),
ww,
wh,
((RND() / 16) & (GrNumColors() - 1)) | GrOR,
bg
    );
}
GrKeyRead();
GrFramedBox(ww/4-1,wh/4-1,ww/4+ww+1,wh/4+wh+1,wdt,&icolors);
drawing(ww/4,wh/4,ww,wh,c,bg);
while(!GrKeyPressed()) {
    drawing(ww/4+(RND()%100),
wh/4+(RND()%100),
ww,
wh,
((RND() / 16) & (GrNumColors() - 1)) | GrAND,
bg
    );
}
GrKeyRead();
```

```
}
```

curstest.c

```
/**
 ** curstest.c ---- test cursors
 **
 ** Copyright (c) 1995 Csaba Biegl, 820 Stirrup Dr, Nashville, TN 37221
 ** [e-mail: csaba@vuse.vanderbilt.edu]
 **
 ** This is a test/demo file of the GRX graphics library.
 ** You can use GRX test/demo files as you want.
 **
 ** The GRX graphics library is free software; you can redistribute it
 ** and/or modify it under some conditions; see the "copying.grx" file
 ** for details.
 **
 ** This library is distributed in the hope that it will be useful,
 ** but WITHOUT ANY WARRANTY; without even the implied warranty of
 ** MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 **
 **/

#include "test.h"  /* see [test.h], page 150 */

char p16d[] = {
    0,1,0,0,0,0,0,0,0,0,0,0,1,1,0,0,
    1,2,1,0,0,0,0,0,0,0,0,1,2,2,1,0,
    1,2,2,1,0,0,0,0,0,0,1,2,0,0,2,1,
    1,2,2,2,1,0,0,0,0,0,1,2,0,0,2,1,
    1,2,2,2,2,1,0,0,0,0,0,1,2,2,1,0,
    1,2,2,2,2,2,1,0,0,0,0,0,1,1,0,0,
    1,2,2,2,2,2,2,1,0,0,0,0,0,0,0,0,
    1,2,2,2,2,2,2,2,1,0,0,0,0,0,0,0,
    1,2,2,2,2,2,2,2,2,1,0,0,0,0,0,0,
    1,2,2,2,2,2,2,2,2,2,1,0,0,0,0,0,
    1,2,2,2,2,2,2,2,2,2,2,1,0,0,0,0,
    1,2,2,2,2,1,1,1,1,1,1,0,0,0,0,0,
    1,2,2,2,1,0,0,0,0,0,0,0,0,0,0,0,
    1,2,2,1,0,0,0,0,0,0,0,0,0,0,0,0,
    1,2,1,0,0,0,0,0,0,0,0,0,0,0,0,0,
    0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
};

TESTFUNC(cursortest)
{
```

```
GrColor bgc = GrAllocColor(0,0,128);
GrColor fgc = GrAllocColor(255,255,0);
GrColor msc[3];
GrCursor *cur;
int x,y;

msc[0] = 2;
msc[1] = GrWhite();
msc[2] = GrAllocColor(255,0,0);
cur = GrBuildCursor(p16d,16,16,16,1,1,msc);
x = GrScreenX() / 2;
y = GrScreenY() / 2;
GrMoveCursor(cur,x,y);
GrClearScreen(bgc);
GrSetColor((GrNumColors() - 1),255,255,255);
drawing(0,0,GrSizeX(),GrSizeY(),fgc,GrNOCOLOR);
GrFilledBox(0,0,320,120,GrAllocColor(0,255,255));
GrTextXY( 10,90,"ANDmask",GrBlack(),GrNOCOLOR);
GrTextXY( 90,90,"ORmask", GrBlack(),GrNOCOLOR);
GrTextXY(170,90,"Save",   GrBlack(),GrNOCOLOR);
GrTextXY(250,90,"Work",   GrBlack(),GrNOCOLOR);
GrDisplayCursor(cur);
for( ; ; ) {
    GrBitBlt(
NULL,10,10,
&cur->work,cur->xwork/2,0,cur->xwork/2+cur->xsize-1,cur->ysize-1,
GrWRITE
    );
    GrBitBlt(
NULL,90,10,
&cur->work,0,0,cur->xsize-1,cur->ysize-1,
GrWRITE
    );
    GrBitBlt(
NULL,170,10,
&cur->work,0,cur->ysize,cur->xwork-1,cur->ysize+cur->ywork-1,
GrWRITE
    );
    GrBitBlt(
NULL,250,10,
&cur->work,0,cur->ysize+cur->ywork,cur->xwork-1,cur->ysize+2*cur->ywork-
1,
GrWRITE
    );
    GrTextXY(0,GrMaxY()-20,"Type u d l r U D L R or q to quit",GrWhite(),GrNOCOLOR);
    switch(GrKeyRead()) {
case 'u': y--; break;
case 'd': y++; break;
case 'l': x--; break;
case 'r': x++; break;
case 'U': y -= 10; break;
case 'D': y += 10; break;
case 'L': x -= 10; break;
case 'R': x += 10; break;
case 'q': return;
default:  continue;
    }
    if(x < 0) x = 0;
    if(x > GrScreenX()) x = GrScreenX();
    if(y < 100) y = 100;
    if(y > GrScreenY()) y = GrScreenY();
    GrMoveCursor(cur,x,y);
}
}
```

demogrx.c

```
/**
 ** demogrx.c ---- GRX Test programs launcher
 **
 ** Copyright (C) 2000,2001 Mariano Alvarez Fernandez
 ** [e-mail: malfer@teleline.es]
 **
 ** This is a test/demo file of the GRX graphics library.
 ** You can use GRX test/demo files as you want.
 **
 ** The GRX graphics library is free software; you can redistribute it
 ** and/or modify it under some conditions; see the "copying.grx" file
 ** for details.
 **
 ** This library is distributed in the hope that it will be useful,
 ** but WITHOUT ANY WARRANTY; without even the implied warranty of
 ** MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 **
 **/

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "grx20.h"
#include "grxkeys.h"
#include "gfaz.h"  /* see [gfaz.h], page 146 */
#include "drawing.h"  /* see [drawing.h], page 146 */
```

```
/* default mode */

static int gwidth = 640;
static int gheight = 480;
static int gbpp = 16;

char *wintitle =
    "GRX 2.4.7, the graphics library";

char *animatedtext =
    "GRX 2.4.7, the graphics library for DJGPPv2, Linux, X11 and Win32";

#define NDEMOS 33

#define ID_ARCTEST   1
#define ID_BB1TEST   2
#define ID_BLITTEST  3
#define ID_CIRCTEST  4
#define ID_CLIPTEST  5
#define ID_COLOROPS  6
#define ID_CURSTEST  7
#define ID_FONTTEST  8
#define ID_IMGTEST   9
#define ID_JPGTEST  10
#define ID_KEYS     11
#define ID_LIFE     12
#define ID_LINETEST 13
#define ID_MOUSETST 14
#define ID_PCIRCTST 15
#define ID_PNMTEST  16
#define ID_PNGTEST  17
#define ID_POLYTEST 18
#define ID_RGBTEST  19
#define ID_SCROLTST 20
#define ID_SBCTEST  21
#define ID_SPEEDTST 22
#define ID_TEXTPATT 23
#define ID_WINCLIP  24
#define ID_WINTEST  25
#define ID_FNTDEMO1 26
#define ID_FNTDEMO2 27
#define ID_FNTDEMO3 28
#define ID_FNTDEMO4 29
#define ID_MODETEST 50
#define ID_PAGE1    81
#define ID_PAGE2    82
```

```
#define ID_EXIT     99

typedef struct {
    int cid;
    char *prog;
    char *text;
} ProgTable;

static ProgTable ptable[NDEMOS] = {
    {ID_ARCTEST, "arctest", "arctest.c -> test arc outline and filled arc drawing"},
    {ID_BB1TEST, "bb1test", "bb1test.c -> test GrBitBlt1bpp routine"},
    {ID_BLITTEST, "blittest", "blittest.c -> test various bitblt-s"},
    {ID_CIRCTEST, "circtest", "circtest.c -> test circle and ellipse rendering"},
    {ID_CLIPTEST, "cliptest", "cliptest.c -> test clipping"},
    {ID_COLOROPS, "colorops", "colorops.c -> test WRITE, XOR, OR, and AND draw modes"}
    {ID_CURSTEST, "curstest", "curstest.c -> test cursors"},
    {ID_FONTTEST, "fonttest", "fonttest.c -> test text drawing"},
    {ID_IMGTEST, "imgtest", "imgtest.c -> test image functions mapping"},
    {ID_JPGTEST, "jpgtest", "jpgtext.c -> text context to jpeg functions"},
    {ID_KEYS, "keys", "keys.c -> test keyboard input"},
    {ID_LIFE, "life", "life.c -> Conway's life program"},
    {ID_LINETEST, "linetest", "linetest.c -> test wide and patterned lines"},
    {ID_MOUSETST, "mousetst", "mousetst.c -> test mouse cursor and mouse/keyboard inpu
    {ID_PCIRCTST, "pcirctst", "pcirctst.c -> test custom circle and el-
lipse rendering"},
    {ID_PNMTEST, "pnmtest", "pnmtext.c -> text context to pnm functions"},
    {ID_PNGTEST, "pngtest", "pngtext.c -> text context to png functions"},
    {ID_POLYTEST, "polytest", "polytest.c -> test polygon rendering"},
    {ID_RGBTEST, "rgbtest", "rgbtest.c -> show 256 color RGB palette"},
    {ID_SBCTEST, "sbctest", "sbctest.c -> test subcontext operations"},
    {ID_SCROLTST, "scroltst", "scroltst.c -> test virtual screen set/scroll"},
    {ID_SPEEDTST, "speedtst", "speedtst.c -> check all available frame drivers speed"}
    {ID_TEXTPATT, "textpatt", "textpatt.c -> test patterned text"},
    {ID_WINCLIP, "winclip", "winclip.c -> clip a drawing to various win-
dows (contexts)"},
    {ID_WINTEST, "wintest", "wintest.c -> test window (context) mapping"},
    {ID_FNTDEMO1, "fontdemo ncen22b.fnt", "fontdemo ncen22b.fnt -> test a GRX font"},
    {ID_FNTDEMO2, "fontdemo ter-114b.res", "fontdemo ter-114b.res -> test a RES font"}
    {ID_FNTDEMO3, "fontdemo ter-114n.fna", "fontdemo ter-114n.fna -> test a FNA font"}
    {ID_FNTDEMO4, "fontdemo ter-114v.psf", "fontdemo ter-114v.psf -> test a PSF font"}
    {ID_MODETEST, "modetest", "modetest.c -> test all available graphics modes"},
    {ID_PAGE1, "", "Change to page 1"},
    {ID_PAGE2, "", "Change to page 2"},
    {ID_EXIT, "", "Exit GRX test programs launcher"}
};

#define PX0 10
```

```
#define PX1 115
#define PX2 220
#define PY0 10
#define PY1 54
#define PY2 98
#define PY3 142
#define PY4 186
#define PY5 230
#define PY6 274
#define PY7 318
#define PY8 362

#define NBUTTONSP1 26

static Button bp1[NBUTTONSP1] = {
    {PX0, PY0, 100, 40, IND_BLUE, IND_YELLOW, "ArcTest", BSTATUS_SELECTED, ID_ARCTEST}
    {PX0, PY1, 100, 40, IND_BLUE, IND_YELLOW, "Bb1Test", 0, ID_BB1TEST},
    {PX0, PY2, 100, 40, IND_BLUE, IND_YELLOW, "BlitTest", 0, ID_BLITTEST},█
    {PX0, PY3, 100, 40, IND_BLUE, IND_YELLOW, "CircTest", 0, ID_CIRCTEST},█
    {PX0, PY4, 100, 40, IND_BLUE, IND_YELLOW, "ClipTest", 0, ID_CLIPTEST},█
    {PX0, PY5, 100, 40, IND_BLUE, IND_YELLOW, "Colorops", 0, ID_COLOROPS},█
    {PX0, PY6, 100, 40, IND_BLUE, IND_YELLOW, "CursTest", 0, ID_CURSTEST},█
    {PX0, PY7, 100, 40, IND_BLUE, IND_YELLOW, "ImgTest", 0, ID_IMGTEST},
    {PX0, PY8, 100, 40, IND_BLUE, IND_YELLOW, "JpgTest", 0, ID_JPGTEST},
    {PX1, PY0, 100, 40, IND_BLUE, IND_YELLOW, "Keys", 0, ID_KEYS},
    {PX1, PY1, 100, 40, IND_BLUE, IND_YELLOW, "Life", 0, ID_LIFE},
    {PX1, PY2, 100, 40, IND_BLUE, IND_YELLOW, "LineTest", 0, ID_LINETEST},█
    {PX1, PY3, 100, 40, IND_BLUE, IND_YELLOW, "MouseTst", 0, ID_MOUSETST},█
    {PX1, PY4, 100, 40, IND_BLUE, IND_YELLOW, "PcircTst", 0, ID_PCIRCTST},█
    {PX1, PY5, 100, 40, IND_BLUE, IND_YELLOW, "PnmTest", 0, ID_PNMTEST},
    {PX1, PY6, 100, 40, IND_BLUE, IND_YELLOW, "PngTest", 0, ID_PNGTEST},
    {PX1, PY7, 100, 40, IND_BLUE, IND_YELLOW, "PolyTest", 0, ID_POLYTEST},█
    {PX1, PY8, 100, 40, IND_BLUE, IND_YELLOW, "RgbTest", 0, ID_RGBTEST},
    {PX2, PY0, 100, 40, IND_BLUE, IND_YELLOW, "SbcTest", 0, ID_SBCTEST},
    {PX2, PY1, 100, 40, IND_BLUE, IND_YELLOW, "ScrolTst", 0, ID_SCROLTST},█
    {PX2, PY2, 100, 40, IND_BLUE, IND_YELLOW, "SpeedTst", 0, ID_SPEEDTST},█
    {PX2, PY3, 100, 40, IND_BLUE, IND_YELLOW, "WinClip", 0, ID_WINCLIP},
    {PX2, PY4, 100, 40, IND_BLUE, IND_YELLOW, "WinTest", 0, ID_WINTEST},
    {PX2, PY6, 100, 40, IND_GREEN, IND_WHITE, "Page 2", 0, ID_PAGE2},
    {PX2, PY7, 100, 40, IND_BROWN, IND_WHITE, "ModeTest", 0, ID_MODETEST},█
    {PX2, PY8, 100, 40, IND_RED, IND_WHITE, "Exit", 0, ID_EXIT}
};

#define NBUTTONSP2  9

static Button bp2[NBUTTONSP2] = {
    {PX0, PY0, 100, 40, IND_BLUE, IND_YELLOW, "FontTest", BSTATUS_SELECTED, ID_FONTTES
    {PX0, PY1, 100, 40, IND_BLUE, IND_YELLOW, "TextPatt", 0, ID_TEXTPATT},
    {PX0, PY2, 100, 40, IND_BLUE, IND_YELLOW, "FontDemo1", 0, ID_FNTDEMO1},
    {PX0, PY3, 100, 40, IND_BLUE, IND_YELLOW, "FontDemo2", 0, ID_FNTDEMO2},
    {PX0, PY4, 100, 40, IND_BLUE, IND_YELLOW, "FontDemo3", 0, ID_FNTDEMO3},
    {PX0, PY5, 100, 40, IND_BLUE, IND_YELLOW, "FontDemo4", 0, ID_FNTDEMO4},
    {PX2, PY6, 100, 40, IND_GREEN, IND_WHITE, "Page 1", 0, ID_PAGE1},
    {PX2, PY7, 100, 40, IND_BROWN, IND_WHITE, "ModeTest", 0, ID_MODETEST},
    {PX2, PY8, 100, 40, IND_RED, IND_WHITE, "Exit", 0, ID_EXIT}
};

static Button_Group bgp1 = { 20, 30, bp1, NBUTTONSP1, 0, 0 };
static Button_Group bgp2 = { 20, 30, bp2, NBUTTONSP2, 0, 0 };
static Button_Group *bgact = &bgp1;

static Board brd =
    { 0, 0, 640, 480, IND_BLACK, IND_CYAN, IND_DARKGRAY, 1 };
static Board brdimg =
    { 384, 46, 235, 157, IND_BLACK, IND_CYAN, IND_DARKGRAY, 1 };

static GrFont *grf_std;
static GrFont *grf_big;
GrTextOption grt_centered;
GrTextOption grt_left;

static GrContext *grcglob = NULL;
static int worg = 0, horg = 0;

/* Internal routines */

static void ini_graphics(void);
static void ini_objects(void);
static void paint_screen(void);
static void the_title(int x, int y);
static void the_info(int x, int y);
static int pev_command(Event * ev);
static int pev_select(Event * ev);
static void paint_foot(char *s);
static void paint_animation(void);
static void disaster(char *s);

/************************************************************************/

int main(int argc, char **argv)
{
    Event ev;
    char buffer[100];
    if (argc >= 4) {
gwidth = atoi(argv[1]);
gheight = atoi(argv[2]);
gbpp = atoi(argv[3]);
    }

    ini_graphics();
    GrSetWindowTitle(wintitle);
    ini_objects();
    paint_screen();

    while (1) {
event_read(&ev);
if (ev.type == EV_MOUSE) {
    ev.p2 -= worg;
    ev.p3 -= horg;
}
if (ev.type == EV_END)
    break;
if ((ev.type == EV_KEY) && (ev.p1 == GrKey_Escape))
    break;
if ((ev.type == EV_KEY) && (ev.p1 == ’s’)) {
    GrSaveContextToPpm(NULL, "demogrx.ppm", "DemoGRX");
    continue;
}
        if (pev_button_group(&ev, bgact))
    continue;
if (pev_command(&ev))
    continue;
if (pev_select(&ev))
    continue;
if (ev.type == EV_MOUSE) {
    if (ev.p1 == MOUSE_LB_PRESSED) {
        sprintf(buffer, "%ld %ld over a button, please",ev.p2,ev.p3);
paint_foot(buffer);
    }
    else if (ev.p1 == MOUSE_LB_RELEASED)
paint_foot("Hold down left mouse buttom to see a comment");
}
if (ev.type == EV_NULL)
    paint_animation();
    }

    gfaz_fin();
    return 0;
}
```

```
/************************************************************************/

static void ini_graphics(void)
{
    gfaz_ini(gwidth, gheight, gbpp);
    gwidth = GrScreenX();
    gheight = GrScreenY();
    grcglob = NULL;
    if (gwidth > 640 || gheight > 480) {
GrClearScreen(GrAllocColor(120, 90, 60));
worg = (gwidth - 640) / 2;
horg = (gheight - 480) / 2;
        grcglob = GrCreateSubContext(worg, horg, worg + 639, horg + 479,
                                     NULL, NULL);
GrSetContext(grcglob);
    }
}

/************************************************************************/

static void ini_objects(void)
{
    grf_std = GrLoadFont("lucb21.fnt");
    if (grf_std == NULL) {
grf_std = GrLoadFont("../fonts/lucb21.fnt");
if (grf_std == NULL)
    disaster("lucb21.fnt not found");
    }

    grf_big = GrLoadFont("lucb40b.fnt");
    if (grf_big == NULL) {
grf_big = GrLoadFont("../fonts/lucb40b.fnt");
if (grf_big == NULL)
    disaster("lucb40b.fnt not found");
    }

    grt_centered.txo_bgcolor.v = GrNOCOLOR;
    grt_centered.txo_direct = GR_TEXT_RIGHT;
    grt_centered.txo_xalign = GR_ALIGN_CENTER;
    grt_centered.txo_yalign = GR_ALIGN_CENTER;
    grt_centered.txo_chrtype = GR_BYTE_TEXT;

    grt_left.txo_bgcolor.v = GrNOCOLOR;
    grt_left.txo_direct = GR_TEXT_RIGHT;
    grt_left.txo_xalign = GR_ALIGN_LEFT;
    grt_left.txo_yalign = GR_ALIGN_CENTER;
    grt_left.txo_chrtype = GR_BYTE_TEXT;
```

```
}

/************************************************************************/

static void paint_screen(void)
{
    GrContext *grc;

    paint_board(&brd);
    paint_button_group(bgact);
    paint_board(&brdimg);
    grc = GrCreateSubContext(brdimg.x + 4, brdimg.y + 4,
     brdimg.x + brdimg.wide - 5,
     brdimg.y + brdimg.high - 5, grcglob, NULL);
    if (bgact == &bgp1)
        GrLoadContextFromPnm(grc, "pnmtest.ppm");
    else
        GrLoadContextFromPnm(grc, "pnmtest2.ppm");
    GrDestroyContext(grc);
    the_info(500, 215);
    drawing(400, 280, 200, 150, BROWN, DARKGRAY);
    the_title(500, 330);
    paint_foot("Hold down left mouse buttom to see a comment");
}

/************************************************************************/

static void the_title(int x, int y)
{
    char *t1 = "GRX 2.4.7";
    char *t2 = "test programs launcher";

    grt_centered.txo_fgcolor.v = LIGHTGREEN;

    grt_centered.txo_font = grf_big;
    GrDrawString(t1, strlen(t1), 0 + x, 0 + y, &grt_centered);

    grt_centered.txo_font = grf_std;
    GrDrawString(t2, strlen(t2), 0 + x, 40 + y, &grt_centered);
}

/************************************************************************/

static void the_info(int x, int y)
{
    char aux[81], sys[4] = "?";
    int nsys;
    grt_centered.txo_fgcolor.v = CYAN;
    grt_centered.txo_font = grf_std;

    nsys = GrGetLibrarySystem();
    if (nsys == GRX_VERSION_TCC_8086_DOS)
strcpy(sys, "TCC");
    if (nsys == GRX_VERSION_GCC_386_DJGPP)
strcpy(sys, "DJ2");
    if (nsys == GRX_VERSION_GCC_386_LINUX)
strcpy(sys, "LNX");
    if (nsys == GRX_VERSION_GCC_386_X11)
strcpy(sys, "X11");
    if (nsys == GRX_VERSION_GCC_X86_64_LINUX)
strcpy(sys, "L64");
    if (nsys == GRX_VERSION_GCC_X86_64_X11)
strcpy(sys, "X64");
    if (nsys == GRX_VERSION_GENERIC_X11)
strcpy(sys, "X11");
    if (nsys == GRX_VERSION_WATCOM_DOS4GW)
strcpy(sys, "WAT");
    if (nsys == GRX_VERSION_GCC_386_WIN32)
strcpy(sys, "W32");

    sprintf(aux, "Version:%x System:%s", GrGetLibraryVersion(), sys);
    GrDrawString(aux, strlen(aux), 0 + x, 0 + y, &grt_centered);

    sprintf(aux, "VideoDriver: %s", GrCurrentVideoDriver()->name);
    GrDrawString(aux, strlen(aux), 0 + x, 25 + y, &grt_centered);

    sprintf(aux, "Mode: %dx%d %d bpp", GrCurrentVideoMode()->width,
    GrCurrentVideoMode()->height, GrCurrentVideoMode()->bpp);
    GrDrawString(aux, strlen(aux), 0 + x, 50 + y, &grt_centered);
}

/***********************************************************************/

static int pev_command(Event * ev)
{
    int i;
    char nprog[81];

    if (ev->type == EV_COMMAND) {
if (ev->p1 == ID_EXIT) {
    par_event_queue(EV_END, 0, 0, 0);
    return 1;
}
```

```
    if (ev->p1 == ID_PAGE1) {
      bgact = &bgp1;
      paint_screen();
  return 1;
}
    if (ev->p1 == ID_PAGE2) {
      bgact = &bgp2;
      paint_screen();
  return 1;
}
for (i = 0; i < NDEMOS; i++) {
  if (ev->p1 == ptable[i].cid) {
gfaz_fin();
#if defined(__MSDOS__) || defined(__WIN32__)
if (ev->p1 == ID_MODETEST)
  strcpy(nprog, "..\\bin\\");
else
  strcpy(nprog, ".\\");
#else
if (ev->p1 == ID_MODETEST)
  strcpy(nprog, "../bin/");
else
  strcpy(nprog, "./");
#endif
#if defined(__XWIN__)
strcat(nprog, "x");
#endif
strcat(nprog, ptable[i].prog);
system(nprog);
ini_graphics();
        GrSetWindowTitle(wintitle);
paint_screen();
return 1;
  }
}
  }
  return 0;
}

/**************************************************************************/

static int pev_select(Event * ev)
{
  int i;

  if (ev->type == EV_SELECT) {
for (i = 0; i < NDEMOS; i++) {
    if (ev->p1 == ptable[i].cid) {
paint_foot(ptable[i].text);
return 1;
    }
}
    }
    return 0;
}

/*************************************************************************/

static void paint_foot(char *s)
{
    grt_centered.txo_fgcolor.v = LIGHTGREEN;
    grt_centered.txo_font = grf_std;

    GrSetClipBox(10, 440, 630, 470);
    GrClearClipBox(CYAN);
    GrDrawString(s, strlen(s), 320, 455, &grt_centered);
    GrResetClipBox();
}

/*************************************************************************/

static void paint_animation(void)
{
    static int pos = 620;
    static int ini = 0;
    static GrContext *grc;
    int ltext, wtext;

    if (!ini) {
grc = GrCreateContext(620, 30, NULL, NULL);
if (grc == NULL)
    return;
ini = 1;
    }

    grt_left.txo_fgcolor.v = CYAN;
    grt_left.txo_font = grf_std;
    ltext = strlen(animatedtext);
    wtext = GrStringWidth(animatedtext, ltext, &grt_left);

    GrSetContext(grc);
    GrClearContext(DARKGRAY);
    GrDrawString(animatedtext, ltext, pos, 15, &grt_left);
    GrSetContext(grcglob);
    GrBitBlt(NULL, 10, 8, grc, 0, 0, 629, 29, GrWRITE);

    pos -= 1;
    if (pos <= -wtext)
pos = 620;
}

/*************************************************************************/

static void disaster(char *s)
{
    void _GrCloseVideoDriver(void);

    char aux[81];

    gfaz_fin();
    _GrCloseVideoDriver();
    printf("DemoGRX: %s\n", s);
    printf("press Return to continue\n");
    fgets(aux, 80, stdin);
    exit(1);
}
```

fontdemo.c

```
/**
 ** fontdemo.c ---- demonstrate a font
 **
 ** Copyright (C) 2002 Dimitar Zhekov
 ** E-Mail: jimmy@is-vn.bg
 **
 ** This is a test/demo file of the GRX graphics library.
 ** You can use GRX test/demo files as you want.
 **
 ** The GRX graphics library is free software; you can redistribute it
 ** and/or modify it under some conditions; see the "copying.grx" file
 ** for details.
 **
 ** This library is distributed in the hope that it will be useful,
 ** but WITHOUT ANY WARRANTY; without even the implied warranty of
 ** MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 **
 **/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```
#include "grx20.h"
#include "grxkeys.h"

static GrTextOption opt;
static int curx = 0, cury = 0;
/* deltax and deltay are the additional columns/lines between characters */
static int deltax = 0, deltay = 0;

static void gnewl(void)
{
cury += GrCharHeight(’A’, &opt) + deltay;
curx = 0;
if(cury + GrCharHeight(’A’, &opt) > GrSizeY() + deltay) {
    if(GrKeyRead() == GrKey_F10) {
GrUnloadFont(opt.txo_font);
exit(0);
    }
    GrClearScreen(opt.txo_bgcolor.v);
    cury = 0;
}
}

/* all control characters are displayed 1:1 */
static void gputc(int c)
{
if(curx + GrCharWidth(c, &opt) + deltax > GrSizeX()) gnewl();
GrDrawChar(c, curx, cury, &opt);
curx += GrCharWidth(c, &opt) + deltax;
}

static void gputs(const char *s)
{
while(*s != ’\0’) gputc((unsigned char) *s++);
gnewl();
}

static void revert(void)
{
GrColor color;

color = opt.txo_fgcolor.v;
opt.txo_fgcolor.v = opt.txo_bgcolor.v | (color & GR_UNDERLINE_TEXT);
opt.txo_bgcolor.v = color & ~GR_UNDERLINE_TEXT;
GrClearScreen(color);
curx = cury = 0;
}
```

```
int main(int argc, char **argv)
{
int i, n;
char *s;
char *bad = NULL;
int c;

int width = 0, height = 0, bpp = 8, gray = 192, attributes = 0;

char *name, *testname;
GrFontHeader *hdr;
FILE *f;
char buffer[0x20];
GrKeyType key;

/* unfortunately not all systems support getopt() */
for(i = 1; i < argc; i++) {
    s = argv[i];
    if(*s != '-' || ((c = *++s) == '\0') || *++s != '\0') break;
    if(c == '-') {
i++;
break;
    }
    if(++i == argc) {
fprintf(stderr, "-%c: argument required\n", c);
return(1);
    }
    if(sscanf(argv[i], "%d", &n) != 1 || n < 0) {
fprintf(stderr, "%s: invalid argument\n", argv[i]);
exit(1);
    }
    switch(c) {
case 'x' : width = n; break;
case 'y' : height = n; break;
case 'b' : if((bpp = n) < 2 || bpp > 32) bad = "bpp"; break;
case 'g' : if((gray = n) > 255) bad = "gray"; break;
case 'X' : if((deltax = n) > 31) bad = "deltax"; break;
case 'Y' : if((deltay = n) > 31) bad = "deltay"; break;
case 'a' : if((attributes = n) > 3) bad = "attributes"; break;
default  : {
    fprintf(stderr, "-%c: invalid option\n", c);
    return(1);
}
    }
    if(bad) {
fprintf(stderr, "%d: %s out of range\n", n, bad);
return(1);
    }
}

if(i == argc) {
    printf(
"usage:\tfontdemo [-x WIDTH] [-y HEIGHT] [-b BPP] [-g COMPONENT]\n"
      "\t[-X DELTAX] [-Y DELTAY] [-a ATTRIBUTES] FONT [FILE...]\n"
    );
    return(0);
}

name = argv[i++];
opt.txo_font = GrLoadFont(name);
        if(opt.txo_font == NULL && (testname = malloc(strlen(name) + 10)) != NULL) {
            /* try again, this is a test and the path can not been set yet */
#if defined(__MSDOS__) || defined(__WIN32__)
            sprintf( testname,"..\\fonts\\%s",name );
#else
            sprintf( testname,"../fonts/%s",name );
#endif
            opt.txo_font = GrLoadFont(testname);
    free(testname);
        }
if(opt.txo_font == NULL) {
    fprintf(stderr, "%s: load error\n", name);
    return(1);
}
hdr = &opt.txo_font->h;

if(height == 0) {
    if(width == 0) {
switch(hdr->height) {
    case 8 :
    case 14 : height = 400; break;
    case 24 : height = 768; break;
    default : height = hdr->height * 30;
}
    }
    else height = width * 3 / 4;
}
if(width == 0) width = height == 400 ? 640 : height * 4 / 3;

GrSetMode(GR_width_height_bpp_graphics, width, height, bpp);
if(!gray || (opt.txo_fgcolor.v = GrAllocColor(gray, gray, gray)) == GrNO-
COLOR) opt.txo_fgcolor.v = GrWhite();
if(attributes & 0x02) opt.txo_fgcolor.v |= GR_UNDERLINE_TEXT;
```

```
opt.txo_bgcolor.v = GrBlack();
if(attributes & 0x01) revert();
opt.txo_chrtype = GR_BYTE_TEXT;
opt.txo_direct = GR_TEXT_RIGHT;
opt.txo_xalign = GR_ALIGN_LEFT;
opt.txo_yalign = GR_ALIGN_TOP;

sprintf(buffer, "%s %dx%d", hdr->name, GrCharWidth('A', &opt), GrCharHeight('A', &opt)
gputs(buffer);
sprintf(buffer, "%dx%d@%lu", GrSizeX(), GrSizeY(), (unsigned long) GrNumColors());
gputs(buffer);
gnewl();

gputs("THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG");
gputs("the quick brown fox jumps over the lazy dog");
gnewl();

if(hdr->minchar <= 0xC0 && hdr->minchar + hdr->numchars >= 0x100) {
    gputs("     ");
    gputs("     ");
    gnewl();
}

/* ascii table, or to be precise, a full table of the current font */
opt.txo_chrtype = GR_WORD_TEXT;
for(c = 0; c < hdr->numchars; c++) {
    gputc(hdr->minchar + c);
    if(c % 0x20 == 0x1F) gnewl();
}
gnewl();
if(c % 0x20 != 0) gnewl();
opt.txo_chrtype = GR_BYTE_TEXT;

while(i < argc) {
    name = argv[i++];
    if((f = fopen(name, "r")) == NULL) {
perror(name);
return(1);
    }
    while((c = getc(f)) != EOF) if(c != '\n') gputc(c); else gnewl();
    if(ferror(f) != 0 || fclose(f) != 0) {
perror(name);
return(1);
    }
}

/* enter and esc are < 0x100 and displayed 1:1 */
gputs("F1-new line  F5-toggle reverse  F7-toggle underline  F10-exit");
gnewl();

while((key = GrKeyRead()) != GrKey_F10) {
    if(key == GrKey_F1) gnewl();
    else if(key == GrKey_F5) revert();
    else if(key == GrKey_F7) opt.txo_fgcolor.v ^= GR_UNDERLINE_TEXT;
    else if(key < 0x100) gputc(key);
}

GrUnloadFont(opt.txo_font);

return(0);
}
```

fonttest.c

```
/**
 ** fonttest.c ---- test text drawing
 **
 ** Copyright (c) 1995 Csaba Biegl, 820 Stirrup Dr, Nashville, TN 37221
 ** [e-mail: csaba@vuse.vanderbilt.edu]
 **
 ** This is a test/demo file of the GRX graphics library.
 ** You can use GRX test/demo files as you want.
 **
 ** The GRX graphics library is free software; you can redistribute it
 ** and/or modify it under some conditions; see the "copying.grx" file
 ** for details.
 **
 ** This library is distributed in the hope that it will be useful,
 ** but WITHOUT ANY WARRANTY; without even the implied warranty of
 ** MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 **
 **/

#include <string.h>
#include "test.h"  /* see [test.h], page 150 */

int  cx;
int  cy;
GrColor c1;
GrColor c2;
GrColor c3;
GrColor c4;

char test_text[] = {
    "QUICK BROWN FOX JUMPS OVER THE LAZY DOG, "
    "quick brown fox jumps over the lazy dog !@#$%^&*()1234567890"
};

void displayfont(GrFont *font,char *text,int len)
{
GrTextOption opt;
int ww,hh;
int bx,by;
int bw,bh;

memset(&opt,0,sizeof(opt));
opt.txo_font   = font;
opt.txo_xalign = GR_ALIGN_LEFT;
opt.txo_yalign = GR_ALIGN_TOP;
GrFilledBox(0,0,GrSizeX(),GrSizeY(),GrBlack());
opt.txo_direct    = GR_TEXT_RIGHT;
opt.txo_fgcolor.v = GrBlack();
opt.txo_bgcolor.v = c1;
ww = GrStringWidth(text,len,&opt);
hh = GrStringHeight(text,len,&opt);
bw = ww+2*hh;
bh = ww;
bx = cx - bw/2;
by = cy - bh/2;
GrDrawString(text,len,bx+hh,by,&opt);
opt.txo_direct    = GR_TEXT_DOWN;
opt.txo_bgcolor.v = c2;
GrDrawString(text,len,bx+bw-hh,by,&opt);
opt.txo_direct    = GR_TEXT_LEFT;
opt.txo_bgcolor.v = c3;
GrDrawString(text,len,bx+bw-ww-hh,by+bh-hh,&opt);
opt.txo_direct    = GR_TEXT_UP;
opt.txo_bgcolor.v = c4;
GrDrawString(text,len,bx,by+bh-ww,&opt);
GrKeyRead();
GrClearClipBox(GrBlack());
opt.txo_direct    = GR_TEXT_RIGHT;
opt.txo_fgcolor.v = c1;
opt.txo_bgcolor.v = GrBlack();
bx = GrSizeX() / 16;
by = GrSizeY() / 16;
bx = (bx + 7) & ~7;
while(by < GrSizeY()) {
    GrDrawString(test_text,strlen(test_text),bx,by,&opt);
    opt.txo_fgcolor.v ^= GR_UNDERLINE_TEXT;
    by += hh;
```

```
}
GrKeyRead();
}

TESTFUNC(fonttest)
{
GrFont *f;
int i;
char buff[100];
cx = GrSizeX() / 2;
cy = GrSizeY() / 2;
c1 = GrAllocColor(100,200,100);
c2 = GrAllocColor(150,150,100);
c3 = GrAllocColor(100,100,200);
c4 = GrAllocColor(100,180,180);
GrBox(GrSizeX()/16 - 2,
    GrSizeY()/16 - 2,
    GrSizeX() - GrSizeX()/16 + 1,
    GrSizeY() - GrSizeY()/16 + 1,
    GrAllocColor(250,100,100)
);
GrSetClipBox(GrSizeX()/16,
    GrSizeY()/16,
    GrSizeX() - GrSizeX()/16 - 1,
    GrSizeY() - GrSizeY()/16 - 1
);
strcpy(buff,"Default GRX font");
displayfont(&GrDefaultFont,buff,strlen(buff));
strcpy(buff,"Default font scaled to 6x10");
displayfont(
    GrBuildConvertedFont(
&GrDefaultFont,
(GR_FONTCVT_SKIPCHARS | GR_FONTCVT_RESIZE),
6,
10,
' ',
'z'
    ),
    buff,
    strlen(buff)
);
strcpy(buff,"Default font scaled to 12x24");
displayfont(
    GrBuildConvertedFont(
&GrDefaultFont,
(GR_FONTCVT_SKIPCHARS | GR_FONTCVT_RESIZE),
12,
24,
' ',
'z'
    ),
    buff,
    strlen(buff)
);
strcpy(buff,"Default font scaled to 18x36");
displayfont(
    GrBuildConvertedFont(
&GrDefaultFont,
(GR_FONTCVT_SKIPCHARS | GR_FONTCVT_RESIZE),
18,
36,
' ',
'z'
    ),
    buff,
    strlen(buff)
);
strcpy(buff,"Default font scaled to 10x20 proportional");
displayfont(
    GrBuildConvertedFont(
&GrDefaultFont,
(GR_FONTCVT_SKIPCHARS | GR_FONTCVT_RESIZE | GR_FONTCVT_PROPORTION),
10,
20,
' ',
'z'
    ),
    buff,
    strlen(buff)
);
strcpy(buff,"Default font scaled to 10x20 bold");
displayfont(
    GrBuildConvertedFont(
&GrDefaultFont,
(GR_FONTCVT_SKIPCHARS | GR_FONTCVT_RESIZE | GR_FONTCVT_BOLDIFY),
10,
20,
' ',
'z'
    ),
    buff,
    strlen(buff)
);
strcpy(buff,"Default font scaled to 10x20 italic");
```

```
displayfont(
    GrBuildConvertedFont(
&GrDefaultFont,
(GR_FONTCVT_SKIPCHARS | GR_FONTCVT_RESIZE | GR_FONTCVT_ITALICIZE),
10,
20,
' ',
'z'
    ),
    buff,
    strlen(buff)
);
for(i = 0; i < Argc; i++) {
    f = GrLoadFont(Argv[i]);
    if(f) {
sprintf(buff,"This is font %s",Argv[i]);
displayfont(f,buff,strlen(buff));
    }
}
}
```

fswwtest.c

```
/**
 ** fswwtest.c ---- test programmed switching
 **                 between full screen and windowed modes
 **                 for the sdl driver
 **
 ** Copyright (C) 2007 Maurice Lombardi
 **
 ** This is a test/demo file of the GRX graphics library.
 ** You can use GRX test/demo files as you want.
 **
 ** The GRX graphics library is free software; you can redistribute it
 ** and/or modify it under some conditions; see the "copying.grx" file
 ** for details.
 **
 ** This library is distributed in the hope that it will be useful,
 ** but WITHOUT ANY WARRANTY; without even the implied warranty of
 ** MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 **
 **/

#include <stdio.h>
#include "grx20.h"
#include "grxkeys.h"
```

```
#include "drawing.h"  /* see [drawing.h], page 146 */

int main()
{
  int x, y, ww, wh, i = 0;
  GrColor c;
  GrContext *w1, *w2, *w3, *w4;

  do {

    if( i ) GrSetDriver( "sdl::ww gw 600 gh 600 nc 256" );
    else GrSetDriver( "sdl::fs gw 600 gh 600 nc 256" );
    i = ! i;

    GrSetMode( GR_default_graphics );

    x = GrSizeX();
    y = GrSizeY();
    ww = (x / 2) - 10;
    wh = (y / 2) - 10;
    w1 = GrCreateSubContext(5,5,ww+4,wh+4,NULL,NULL);
    w2 = GrCreateSubContext(15+ww,5,ww+ww+14,wh+4,NULL,NULL);
    w3 = GrCreateSubContext(5,15+wh,ww+4,wh+wh+14,NULL,NULL);
    w4 = GrCreateSubContext(15+ww,15+wh,ww+ww+14,wh+wh+14,NULL,NULL);

    GrSetContext(w1);
    c = GrAllocColor(200,100,100);
    drawing(0,0,ww,wh,c,GrBlack());
    GrBox(0,0,ww-1,wh-1,c);

    GrSetContext(w2);
    c = GrAllocColor(100,200,200);
    drawing(0,0,ww,wh,c,GrBlack());
    GrBox(0,0,ww-1,wh-1,c);

    GrSetContext(w3);
    c = GrAllocColor(200,200,0);
    drawing(0,0,ww,wh,c,GrBlack());
    GrBox(0,0,ww-1,wh-1,c);

    GrSetContext(w4);
    c = GrAllocColor(0,100,200);
    drawing(0,0,ww,wh,c,GrBlack());
    GrBox(0,0,ww-1,wh-1,c);

    GrSetContext( NULL );
    GrTextXY(10,wh,"press any key to toggle full screen / windowed modes, es-
cape to end",GrWhite(),GrBlack());

  } while ( GrKeyRead() != GrKey_Escape );
  return 0;
}
```

gfaz.c

```
/**
 ** gfaz.c ---- Mini GUI for GRX
 **
 ** Copyright (C) 2000,2001 Mariano Alvarez Fernandez
 ** [e-mail: malfer@teleline.es]
 **
 ** This is a test/demo file of the GRX graphics library.
 ** You can use GRX test/demo files as you want.
 **
 ** The GRX graphics library is free software; you can redistribute it
 ** and/or modify it under some conditions; see the "copying.grx" file
 ** for details.
 **
 ** This library is distributed in the hope that it will be useful,
 ** but WITHOUT ANY WARRANTY; without even the implied warranty of
 ** MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 **
 **/

#include <stdlib.h>
#include <string.h>
#include <grx20.h>
#include <grxkeys.h>
#include "gfaz.h"  /* see [gfaz.h], page 146 */

GrColor *egacolors;

static int mouse_found = 0;
static int mouse_count = 0;

#define MAX_EVQUEUE 10

static Event evqueue[MAX_EVQUEUE];
static int num_evqueue = 0;

static void (*hook_input_event)( Event * ) = NULL;

/* Internal routines */
```

```
static int read_input( void );
static void input_event_queue( Event *ev );
static int coord_into( int x, int y, int xo, int yo, int xl, int yl );

/**************************************************************************/

int gfaz_ini( int width, int height, int bpp )
{
/*  __djgpp_set_ctrl_c( 0 );*/
/*  _go32_want_ctrl_break( 1 );*/
/*  GrSetMode( GR_default_graphics );*/

  GrSetMode( GR_width_height_bpp_graphics,width,height,bpp );

  egacolors = GrAllocEgaColors();

  if( GrMouseDetect() ){
    mouse_found = 1;
    GrMouseInit();
    GrMouseSetColors( GrWhite(),GrBlack() );
    show_mouse();
    }
  GrMouseEventEnable( 1,mouse_found );

  return 0;
}

/**************************************************************************/

int gfaz_fin( void )
{
  if( mouse_found ){
    hide_mouse();
    GrMouseUnInit();
    }

  GrSetMode( GR_default_text );

  return 0;
}

/**************************************************************************/

void event_read( Event *ev )
{
  while( 1 ){
    if( num_evqueue > 0 ){
      num_evqueue--;
      *ev = evqueue[num_evqueue];
      return;
      }
    if( read_input() ){
      continue;
      }
    ev->type = EV_NULL;
    ev->p1 = 0;
    ev->p2 = 0;
    ev->p3 = 0;
    return;
    }
}

/**************************************************************************/

void event_wait( Event *ev )
{
  while( 1 ){
    event_read( ev );
    if( ev->type != EV_NULL )
      return;
    }
}

/**************************************************************************/

void event_queue( Event *ev )
{
  if( num_evqueue < MAX_EVQUEUE )
    evqueue[num_evqueue++] = *ev;
}

/**************************************************************************/

void par_event_queue( int type, long p1, long p2, long p3 )
{
  Event ev;

  ev.type = type;
  ev.p1 = p1;
  ev.p2 = p2;
  ev.p3 = p3;
  event_queue( &ev );
}
```

```
/****************************************************************************/

void set_hook_input_event( void (*fn)( Event * ) )
{
  hook_input_event = fn;
}

/****************************************************************************/

static void input_event_queue( Event *ev )
{
  if( hook_input_event != NULL )
    hook_input_event( ev );
  else
    event_queue( ev );
}

/**************************************************************************/

static int read_input( void )
{
  GrMouseEvent evt;
  Event evaux;

  GrMouseGetEventT( GR_M_EVENT,&evt,10L );
  if( evt.dtime == -1 ) return 0;

  if( evt.flags & GR_M_KEYPRESS ){
    evaux.type = EV_KEY;
    evaux.p1 = evt.key;
    evaux.p2 = 0;
    evaux.p3 = 0;
    input_event_queue( &evaux );
    }
  evaux.type = EV_MOUSE;
  evaux.p2 = evt.x;
  evaux.p3 = evt.y;
  if( evt.flags & GR_M_LEFT_DOWN ){
    evaux.p1 = MOUSE_LB_PRESSED;
    input_event_queue( &evaux );
    }
  if( evt.flags & GR_M_RIGHT_DOWN ){
    evaux.p1 = MOUSE_RB_PRESSED;
    input_event_queue( &evaux );
    }
  if( evt.flags & GR_M_LEFT_UP ){
    evaux.p1 = MOUSE_LB_RELEASED;
    input_event_queue( &evaux );
    }
  if( evt.flags & GR_M_RIGHT_UP ){
    evaux.p1 = MOUSE_RB_RELEASED;
    input_event_queue( &evaux );
    }
  return 1;
}

/***************************************************************************/

void show_mouse( void )
{
  if( (mouse_count == 0) && mouse_found )
    GrMouseDisplayCursor();

  mouse_count++;
}

/*************************************************************************/

void hide_mouse( void )
{
  mouse_count--;

  if( (mouse_count == 0) && mouse_found )
    GrMouseEraseCursor();
}

/***************************************************************************/

void dboton( int x, int y, int an, int al,
             GrColor c, GrColor ct, char * s, int t )

//   x, y posicin de la esquina izquierda
//   an, al ancho y alto
//   c, ct color del fondo y del texto
//   t, tipo bit 0 0=normal, 1=apretado
//           bit 1 0=no selec, 1=seleccionado

{
  int pol[7][2], prof, pulsd, selec, despl;
  GrTextOption grt;
  GrLineOption glo;
  int mouseblock;
```

```
prof = (t & 0x1) ? 2 : 4;
pulsd = (t & 0x1) ? 1 : 0;
selec = (t & 0x2) ? 1 : 0;
despl = (t & 0x1) ? 1 : 0;

mouseblock = GrMouseBlock( NULL,x,y,x+an-1,y+al-1 );
GrBox( x,y,x+an-1,y+al-1,BLACK );
x = x + 1; y = y + 1;
an = an - 2; al = al - 2;

pol[0][0] = x;                pol[0][1] = y;
pol[1][0] = x + an - 1;       pol[1][1] = y;
pol[2][0] = x + an - 2 - prof; pol[2][1] = y + 1 + prof;
pol[3][0] = x + 1 + prof;     pol[3][1] = y + 1 + prof;
pol[4][0] = x + 1 + prof;     pol[4][1] = y + al - 2 - prof;
pol[5][0] = x;                pol[5][1] = y + al - 1;
pol[6][0] = pol[0][0];        pol[6][1] = pol[0][1];
GrFilledPolygon( 7,pol,pulsd ? DARKGRAY : LIGHTGRAY );
GrPolygon( 7,pol,BLACK );
GrLine( pol[0][0],pol[0][1],pol[3][0],pol[3][1],BLACK );
pol[0][0] = x + an - 1;       pol[0][1] = y + al - 1;
pol[3][0] = x + an - 2 - prof; pol[3][1] = y + al - 2 - prof;
pol[6][0] = pol[0][0];        pol[6][1] = pol[0][1];
GrFilledPolygon( 7,pol,pulsd ? LIGHTGRAY : DARKGRAY );
GrPolygon( 7,pol,BLACK );
GrLine( pol[0][0],pol[0][1],pol[3][0],pol[3][1],BLACK );
GrFilledBox( x+2+prof,y+2+prof,x+an-3-prof,y+al-3-prof,c );

grt.txo_font = &GrFont_PC8x14;
grt.txo_fgcolor.v = ct;
grt.txo_bgcolor.v = GrNOCOLOR;
grt.txo_direct = GR_TEXT_RIGHT;
grt.txo_xalign = GR_ALIGN_CENTER;
grt.txo_yalign = GR_ALIGN_CENTER;
grt.txo_chrtype = GR_BYTE_TEXT;
if( despl )
  GrDrawString( s,strlen( s ),x+an/2+1,y+al/2+1,&grt );
else
  GrDrawString( s,strlen( s ),x+an/2,y+al/2,&grt );

if( selec ){
  glo.lno_color = ct;
  glo.lno_width = 1;
  glo.lno_pattlen = 2;
  glo.lno_dashpat = "\2\1";
  GrCustomBox( x+8,y+al/2-6,x+an-8,y+al/2+5,&glo );
  }
  GrMouseUnBlock( mouseblock );
}

/**************************************************************************/

void paint_button( int x, int y, Button *b )
{
  dboton( x+b->x,y+b->y,b->wide,b->high,
          egacolors[b->tbcolor],egacolors[b->tfcolor],
          b->text,b->status );
}

/**************************************************************************/

void paint_button_group( Button_Group *bg )
{
  int i;

  for( i=0; i<bg->nb; i++ )
    paint_button( bg->x,bg->y,&(bg->b[i]) );
}

/**************************************************************************/

int pev_button_group( Event *ev, Button_Group *bg )
{
  int i;

  if( ev->type == EV_MOUSE ){
    if( ev->p1 == MOUSE_LB_PRESSED ){
      for( i=0; i<bg->nb; i++ )
        if( coord_into( ev->p2,ev->p3,
                        bg->x+bg->b[i].x,bg->y+bg->b[i].y,
                        bg->b[i].wide,bg->b[i].high ) ){
          bg->b[bg->pb].status &= ~BSTATUS_SELECTED;
          paint_button( bg->x,bg->y,&(bg->b[bg->pb]) );
          bg->b[i].status |= BSTATUS_PRESSED | BSTATUS_SELECTED;
          paint_button( bg->x,bg->y,&(bg->b[i]) );
          bg->pb = i;
          bg->abp = 1;
          par_event_queue( EV_SELECT,bg->b[i].bid,0,0 );
          return 1;
          }
      }
    if( ev->p1 == MOUSE_LB_RELEASED ){
      if( bg->abp ){
        i = bg->pb;
        bg->b[i].status &= ~BSTATUS_PRESSED;
        paint_button( bg->x,bg->y,&(bg->b[i]) );
        bg->abp = 0;
        if( coord_into( ev->p2,ev->p3,
                        bg->x+bg->b[i].x,bg->y+bg->b[i].y,
                        bg->b[i].wide,bg->b[i].high ) ){
          par_event_queue( EV_COMMAND,bg->b[i].bid,0,0 );
          }
        return 1;
        }
      }
    }
  else if( ev->type == EV_KEY ){
    if( ev->p1 == GrKey_Right ||
        ev->p1 == GrKey_Down ||
        ev->p1 == GrKey_Tab ){
      if( bg->pb < bg->nb-1 ){
        bg->b[bg->pb].status &= ~BSTATUS_SELECTED;
        paint_button( bg->x,bg->y,&(bg->b[bg->pb]) );
        bg->pb++;
        bg->b[bg->pb].status |= BSTATUS_SELECTED;
        paint_button( bg->x,bg->y,&(bg->b[bg->pb]) );
        par_event_queue( EV_SELECT,bg->b[bg->pb].bid,0,0 );
        }
      return 1;
      }
    else if( ev->p1 == GrKey_Left ||
             ev->p1 == GrKey_Up ||
             ev->p1 == GrKey_BackTab ){
      if( bg->pb > 0 ){
        bg->b[bg->pb].status &= ~BSTATUS_SELECTED;
        paint_button( bg->x,bg->y,&(bg->b[bg->pb]) );
        bg->pb--;
        bg->b[bg->pb].status |= BSTATUS_SELECTED;
        paint_button( bg->x,bg->y,&(bg->b[bg->pb]) );
        par_event_queue( EV_SELECT,bg->b[bg->pb].bid,0,0 );
        }
      return 1;
      }
    else if( ev->p1 == GrKey_Return ){
      par_event_queue( EV_COMMAND,bg->b[bg->pb].bid,0,0 );
      return 1;
      }
    }

  return 0;
```

```
}

/**************************************************************************/

static int coord_into( int x, int y, int xo, int yo, int xl, int yl )
{
  if( x < xo ) return 0;
  if( x >= xo+xl ) return 0;
  if( y < yo ) return 0;
  if( y >= yo+yl ) return 0;
  return 1;
}

/**************************************************************************/

void paint_board( Board *b )
{
  int x1, y1, x2, y2;

  x1 = b->x;
  y1 = b->y;
  x2 = b->x + b->wide - 1;
  y2 = b->y + b->high - 1;

  GrBox( x1,y1,x2,y2,egacolors[b->lcolor] );
  GrBox( x1+1,y1+1,x2-1,y2-1,egacolors[b->bcolor] );
  GrBox( x1+2,y1+2,x2-2,y2-2,egacolors[b->bcolor] );
  GrBox( x1+3,y1+3,x2-3,y2-3,egacolors[b->lcolor] );
  GrFilledBox( x1+4,y1+4,x2-4,y2-4,egacolors[b->color] );
}
```

imgtest.c

```
/**
 ** imgtest.c ---- test image functions mapping
 **
 ** Copyright (c) 1998 Hartmut Schirmer
 **
 ** This is a test/demo file of the GRX graphics library.
 ** You can use GRX test/demo files as you want.
 **
 ** The GRX graphics library is free software; you can redistribute it
 ** and/or modify it under some conditions; see the "copying.grx" file
 ** for details.
 **
 ** This library is distributed in the hope that it will be useful,
 ** but WITHOUT ANY WARRANTY; without even the implied warranty of
 ** MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 **
 **/

#include "test.h"  /* see [test.h], page 150 */

#define PARTS 4

TESTFUNC(imgtest)
{
int  x = GrSizeX();
int  y = GrSizeY();
int  ww = (x / PARTS)-1;
int  wh = (y / PARTS)-1;
int m1, m2, d1, d2;
GrColor c1, c2, c3;
GrContext ctx;
GrImage *img1;
GrImage *img2;
if (! GrCreateContext(ww,wh,NULL,&ctx)) return;

GrSetContext(&ctx);
c1 = GrAllocColor(255,100,0);
c2 = GrAllocColor(0,0,(GrNumColors() >= 16 ? 180 : 63));
c3 = GrAllocColor(0,255,0);
drawing(0,0,ww,wh,c1,c2);
GrBox(0,0,ww-1,wh-1,c1);

GrSetContext(NULL);

img1 = GrImageFromContext(&ctx);
if (!img1) return;

GrFilledBox(0,0,ww+1,wh+1,c3);
GrFilledBox(ww+15,0,2*ww+16,wh+1,c3);

GrBitBlt(NULL,1,1,&ctx,0,0,ww-1,wh-1,0);
GrImageDisplay(ww+16,1,img1);
GrImageDisplayExt(0,wh+4,x-1,y-1, img1);

GrKeyRead();

for (m1=1; m1 <= PARTS ; m1<<=1) {
  for (d1=1; d1 <= PARTS; d1 <<= 1) {
    for (m2=1; m2 <= PARTS ; m2<<=1) {
      for (d2=1; d2 <= PARTS; d2 <<= 1) {
```

```
img2 = GrImageStretch(img1,(m1*ww)/d1, (m2*wh)/d2);
if (img2) {
  GrImageDisplayExt(0,0,x-1,y-1,img2);
  GrImageDestroy(img2);
}
      }
    }
  }
}
GrKeyRead();

/* let's finish with some GrGetScanline / GrPutScanline tests */
for (d1 = 1; d1 < 32; ++d1) {
  for (m1 = wh; m1 < y-wh-d1-1; ++m1) {
    const GrColor *cp;
    cp = GrGetScanline(ww+1,x-ww-d1,m1+1);
    if (cp) {
      GrPutScanline(ww,x-ww-d1-1,m1,cp,GrIMAGE|c2);
    }
  }
}

GrKeyRead();
}
```

jpgtest.c

```
/**
 ** jpgtest.c ---- test the ctx2jpeg routines
 **
 ** Copyright (c) 2001 Mariano Alvarez Fernandez
 ** [e-mail: malfer@teleline.es]
 **
 ** This is a test/demo file of the GRX graphics library.
 ** You can use GRX test/demo files as you want.
 **
 ** The GRX graphics library is free software; you can redistribute it
 ** and/or modify it under some conditions; see the "copying.grx" file
 ** for details.
 **
 ** This library is distributed in the hope that it will be useful,
 ** but WITHOUT ANY WARRANTY; without even the implied warranty of
 ** MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 **/

#include <stdio.h>
```

```
#include <stdlib.h>
#include "grx20.h"
#include "grxkeys.h"

void imagen( char *nf, int scale )
{
  GrContext *grc;
  int wide, high;
  char s[81];
  int w, h;

  GrQueryJpeg( nf,&w,&h );
  sprintf( s,"%s %dx%d scale 1/%d",nf,w,h,scale );
  wide = (w/scale > 600) ? 600 : w/scale;
  high = (h/scale > 400) ? 400 : h/scale;
  GrClearScreen( GrAllocColor( 0,0,200 ) );

  GrBox( 10,40,10+wide+1,40+high+1,GrWhite() );
  grc = GrCreateSubContext( 11,41,11+wide-1,41+high-1,NULL,NULL );
  GrLoadContextFromJpeg( grc,nf,scale );
  GrDestroyContext( grc );

  GrTextXY( 10,10,s,GrBlack(),GrWhite() );
  GrTextXY( 10,50+high,"Press any key to continue",GrBlack(),GrWhite() );
  GrKeyRead();
}

void nojpegsupport( void )
{
  char *s[6] = {
    "Warning!",
    "You need libjpeg (http://www.ijg.org) and enable",
    "jpeg support in the GRX lib (edit makedefs.grx)",
    "to run this demo",
    " ",
    "Press any key to continue..." };
  int i;

  GrClearScreen( GrAllocColor( 0,0,100 ) );
  for( i=0; i<6; i++ )
    GrTextXY( 90,160+i*18,s[i],GrWhite(),GrNOCOLOR );
  GrKeyRead();
}

int main()
{
  GrContext *grc;
```

```
GrSetMode( GR_width_height_bpp_graphics,640,480,24 );

if( !GrJpegSupport() ){
  nojpegsupport();
  GrSetMode(GR_default_text);
  exit( 1 );
  }

imagen( "jpeg1.jpg",1 );
imagen( "jpeg1.jpg",2 );
imagen( "jpeg1.jpg",4 );
imagen( "jpeg1.jpg",8 );
imagen( "jpeg2.jpg",1 );
imagen( "jpeg2.jpg",2 );
imagen( "jpeg2.jpg",4 );
imagen( "jpeg2.jpg",8 );

GrClearScreen( GrAllocColor( 0,100,0 ) );
grc = GrCreateSubContext( 10,40,10+400-1,40+300-1,NULL,NULL );
GrLoadContextFromJpeg( grc,"jpeg1.jpg",2 );
GrDestroyContext( grc );
grc = GrCreateSubContext( 210,150,210+400-1,150+300-1,NULL,NULL );
GrLoadContextFromJpeg( grc,"jpeg2.jpg",2 );
GrDestroyContext( grc );

GrTextXY( 10,10,"Press any key to save color and gray screen",
  GrBlack(),GrWhite() );
GrKeyRead();

GrSaveContextToJpeg( NULL,"p.jpg",75 );
GrSaveContextToGrayJpeg( NULL,"pgray.jpg",75 );

GrClearScreen( GrBlack() );
GrTextXY( 10,10,"Press any key to reload color screen       ",
  GrBlack(),GrWhite() );
GrKeyRead();
GrLoadContextFromJpeg( NULL,"p.jpg",1 );

GrTextXY( 10,10,"Press any key to reload gray screen        ",
  GrBlack(),GrWhite() );
GrKeyRead();
GrClearScreen( GrBlack() );
GrLoadContextFromJpeg( NULL,"pgray.jpg",1 );

GrTextXY( 10,10,"Press any key to end                       ",
  GrBlack(),GrWhite() );
  GrKeyRead();

  GrSetMode(GR_default_text);
  return 0;
}
```

keys.c

```
/**
 **
 ** This is a test/demo file of the GRX graphics library.
 ** You can use GRX test/demo files as you want.
 **
 ** The GRX graphics library is free software; you can redistribute it
 ** and/or modify it under some conditions; see the "copying.grx" file
 ** for details.
 **
 ** This library is distributed in the hope that it will be useful,
 ** but WITHOUT ANY WARRANTY; without even the implied warranty of
 ** MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 **
 ** Changes by D.Zhekov (jimmy@is-vn.bg) 16/02/2004
 **   - converted to a standard test program [GRX I/O only].
 **
 **/

#include <stdio.h>
#include <math.h>
#include <ctype.h>
#include <stdarg.h>
#include "test.h"  /* see [test.h], page 150 */
#include "grxkeys.h"

#if defined(PENTIUM_CLOCK) && (!defined(__GNUC__) || !defined(__i386__))
#undef PENTIUM_CLOCK
#endif

#ifdef PENTIUM_CLOCK
/*******************************************************************************
** This is modified version of keys.c that checks also work of GrKeyPressed()
** and measures time spent in this procedure (at first set divider to clock
** frequency of Your CPU (I have 90MHz): gcc -DPENTIUM_CLOCK=90.
**    A.Pavenis (pavenis@acad.latnet.lv)
*******************  Some time measurements stuff  ***************************
** Macrodefinition RDTSC reads Pentium CPU timestamp counter. It is counting
** CPU clocks. Attempt to use it under 386 or 486 will cause an invalid
** instruction
```

```
*/
#define RDTSC(h,l) __asm__ ("rdtsc" : "=a"(l) , "=d"(h))
inline  long   rdtsc(void)  { long h,l; RDTSC(h,l); return l; }
/* ***************************************************************************/
#endif /* PENTIUM_CLOCK */

#ifdef __DJGPP__
#include <conio.h>
#include <pc.h>
#else
extern int getch(void);
extern int getkey(void);
extern int kbhit(void);
#endif

#define ISPRINT(k) (((unsigned int)(k)) <= 255 && isprint(k))

typedef struct { GrKeyType key;
 char     *name; } KeyEntry;

static KeyEntry Keys[] = {
  { GrKey_NoKey               , "GrKey_NoKey" },
  { GrKey_OutsideValidRange   , "GrKey_OutsideValidRange" },
  { GrKey_Control_A           , "GrKey_Control_A" },
  { GrKey_Control_B           , "GrKey_Control_B" },
  { GrKey_Control_C           , "GrKey_Control_C" },
  { GrKey_Control_D           , "GrKey_Control_D" },
  { GrKey_Control_E           , "GrKey_Control_E" },
  { GrKey_Control_F           , "GrKey_Control_F" },
  { GrKey_Control_G           , "GrKey_Control_G" },
  { GrKey_Control_H           , "GrKey_Control_H" },
  { GrKey_Control_I           , "GrKey_Control_I" },
  { GrKey_Control_J           , "GrKey_Control_J" },
  { GrKey_Control_K           , "GrKey_Control_K" },
  { GrKey_Control_L           , "GrKey_Control_L" },
  { GrKey_Control_M           , "GrKey_Control_M" },
  { GrKey_Control_N           , "GrKey_Control_N" },
  { GrKey_Control_O           , "GrKey_Control_O" },
  { GrKey_Control_P           , "GrKey_Control_P" },
  { GrKey_Control_Q           , "GrKey_Control_Q" },
  { GrKey_Control_R           , "GrKey_Control_R" },
  { GrKey_Control_S           , "GrKey_Control_S" },
  { GrKey_Control_T           , "GrKey_Control_T" },
  { GrKey_Control_U           , "GrKey_Control_U" },
  { GrKey_Control_V           , "GrKey_Control_V" },
  { GrKey_Control_W           , "GrKey_Control_W" },
  { GrKey_Control_X           , "GrKey_Control_X" },
  { GrKey_Control_Y           , "GrKey_Control_Y" },
  { GrKey_Control_Z           , "GrKey_Control_Z" },
  { GrKey_Control_LBracket    , "GrKey_Control_LBracket" },
  { GrKey_Control_BackSlash   , "GrKey_Control_BackSlash" },
  { GrKey_Control_RBracket    , "GrKey_Control_RBracket" },
  { GrKey_Control_Caret       , "GrKey_Control_Caret" },
  { GrKey_Control_Underscore  , "GrKey_Control_Underscore" },
  { GrKey_Space               , "GrKey_Space" },
  { GrKey_ExclamationPoint    , "GrKey_ExclamationPoint" },
  { GrKey_DoubleQuote         , "GrKey_DoubleQuote" },
  { GrKey_Hash                , "GrKey_Hash" },
  { GrKey_Dollar              , "GrKey_Dollar" },
  { GrKey_Percent             , "GrKey_Percent" },
  { GrKey_Ampersand           , "GrKey_Ampersand" },
  { GrKey_Quote               , "GrKey_Quote" },
  { GrKey_LParen              , "GrKey_LParen" },
  { GrKey_RParen              , "GrKey_RParen" },
  { GrKey_Star                , "GrKey_Star" },
  { GrKey_Plus                , "GrKey_Plus" },
  { GrKey_Comma               , "GrKey_Comma" },
  { GrKey_Dash                , "GrKey_Dash" },
  { GrKey_Period              , "GrKey_Period" },
  { GrKey_Slash               , "GrKey_Slash" },
  { GrKey_0                   , "GrKey_0" },
  { GrKey_1                   , "GrKey_1" },
  { GrKey_2                   , "GrKey_2" },
  { GrKey_3                   , "GrKey_3" },
  { GrKey_4                   , "GrKey_4" },
  { GrKey_5                   , "GrKey_5" },
  { GrKey_6                   , "GrKey_6" },
  { GrKey_7                   , "GrKey_7" },
  { GrKey_8                   , "GrKey_8" },
  { GrKey_9                   , "GrKey_9" },
  { GrKey_Colon               , "GrKey_Colon" },
  { GrKey_SemiColon           , "GrKey_SemiColon" },
  { GrKey_LAngle              , "GrKey_LAngle" },
  { GrKey_Equals              , "GrKey_Equals" },
  { GrKey_RAngle              , "GrKey_RAngle" },
  { GrKey_QuestionMark        , "GrKey_QuestionMark" },
  { GrKey_At                  , "GrKey_At" },
  { GrKey_A                   , "GrKey_A" },
  { GrKey_B                   , "GrKey_B" },
  { GrKey_C                   , "GrKey_C" },
  { GrKey_D                   , "GrKey_D" },
  { GrKey_E                   , "GrKey_E" },
  { GrKey_F                   , "GrKey_F" },
  { GrKey_G                   , "GrKey_G" },
  { GrKey_H                  , "GrKey_H" },
  { GrKey_I                  , "GrKey_I" },
  { GrKey_J                  , "GrKey_J" },
  { GrKey_K                  , "GrKey_K" },
  { GrKey_L                  , "GrKey_L" },
  { GrKey_M                  , "GrKey_M" },
  { GrKey_N                  , "GrKey_N" },
  { GrKey_O                  , "GrKey_O" },
  { GrKey_P                  , "GrKey_P" },
  { GrKey_Q                  , "GrKey_Q" },
  { GrKey_R                  , "GrKey_R" },
  { GrKey_S                  , "GrKey_S" },
  { GrKey_T                  , "GrKey_T" },
  { GrKey_U                  , "GrKey_U" },
  { GrKey_V                  , "GrKey_V" },
  { GrKey_W                  , "GrKey_W" },
  { GrKey_X                  , "GrKey_X" },
  { GrKey_Y                  , "GrKey_Y" },
  { GrKey_Z                  , "GrKey_Z" },
  { GrKey_LBracket           , "GrKey_LBracket" },
  { GrKey_BackSlash          , "GrKey_BackSlash" },
  { GrKey_RBracket           , "GrKey_RBracket" },
  { GrKey_Caret              , "GrKey_Caret" },
  { GrKey_UnderScore         , "GrKey_UnderScore" },
  { GrKey_BackQuote          , "GrKey_BackQuote" },
  { GrKey_a                  , "GrKey_a" },
  { GrKey_b                  , "GrKey_b" },
  { GrKey_c                  , "GrKey_c" },
  { GrKey_d                  , "GrKey_d" },
  { GrKey_e                  , "GrKey_e" },
  { GrKey_f                  , "GrKey_f" },
  { GrKey_g                  , "GrKey_g" },
  { GrKey_h                  , "GrKey_h" },
  { GrKey_i                  , "GrKey_i" },
  { GrKey_j                  , "GrKey_j" },
  { GrKey_k                  , "GrKey_k" },
  { GrKey_l                  , "GrKey_l" },
  { GrKey_m                  , "GrKey_m" },
  { GrKey_n                  , "GrKey_n" },
  { GrKey_o                  , "GrKey_o" },
  { GrKey_p                  , "GrKey_p" },
  { GrKey_q                  , "GrKey_q" },
  { GrKey_r                  , "GrKey_r" },
  { GrKey_s                  , "GrKey_s" },
  { GrKey_t                  , "GrKey_t" },
  { GrKey_u                  , "GrKey_u" },
  { GrKey_v                  , "GrKey_v" },
  { GrKey_w                   , "GrKey_w" },
  { GrKey_x                   , "GrKey_x" },
  { GrKey_y                   , "GrKey_y" },
  { GrKey_z                   , "GrKey_z" },
  { GrKey_LBrace              , "GrKey_LBrace" },
  { GrKey_Pipe                , "GrKey_Pipe" },
  { GrKey_RBrace              , "GrKey_RBrace" },
  { GrKey_Tilde               , "GrKey_Tilde" },
  { GrKey_Control_Backspace   , "GrKey_Control_Backspace" },
  { GrKey_Alt_Escape          , "GrKey_Alt_Escape" },
  { GrKey_Control_At          , "GrKey_Control_At" },
  { GrKey_Alt_Backspace       , "GrKey_Alt_Backspace" },
  { GrKey_BackTab             , "GrKey_BackTab" },
  { GrKey_Alt_Q               , "GrKey_Alt_Q" },
  { GrKey_Alt_W               , "GrKey_Alt_W" },
  { GrKey_Alt_E               , "GrKey_Alt_E" },
  { GrKey_Alt_R               , "GrKey_Alt_R" },
  { GrKey_Alt_T               , "GrKey_Alt_T" },
  { GrKey_Alt_Y               , "GrKey_Alt_Y" },
  { GrKey_Alt_U               , "GrKey_Alt_U" },
  { GrKey_Alt_I               , "GrKey_Alt_I" },
  { GrKey_Alt_O               , "GrKey_Alt_O" },
  { GrKey_Alt_P               , "GrKey_Alt_P" },
  { GrKey_Alt_LBracket        , "GrKey_Alt_LBracket" },
  { GrKey_Alt_RBracket        , "GrKey_Alt_RBracket" },
  { GrKey_Alt_Return          , "GrKey_Alt_Return" },
  { GrKey_Alt_A               , "GrKey_Alt_A" },
  { GrKey_Alt_S               , "GrKey_Alt_S" },
  { GrKey_Alt_D               , "GrKey_Alt_D" },
  { GrKey_Alt_F               , "GrKey_Alt_F" },
  { GrKey_Alt_G               , "GrKey_Alt_G" },
  { GrKey_Alt_H               , "GrKey_Alt_H" },
  { GrKey_Alt_J               , "GrKey_Alt_J" },
  { GrKey_Alt_K               , "GrKey_Alt_K" },
  { GrKey_Alt_L               , "GrKey_Alt_L" },
  { GrKey_Alt_Semicolon       , "GrKey_Alt_Semicolon" },
  { GrKey_Alt_Quote           , "GrKey_Alt_Quote" },
  { GrKey_Alt_Backquote       , "GrKey_Alt_Backquote" },
  { GrKey_Alt_Backslash       , "GrKey_Alt_Backslash" },
  { GrKey_Alt_Z               , "GrKey_Alt_Z" },
  { GrKey_Alt_X               , "GrKey_Alt_X" },
  { GrKey_Alt_C               , "GrKey_Alt_C" },
  { GrKey_Alt_V               , "GrKey_Alt_V" },
  { GrKey_Alt_B               , "GrKey_Alt_B" },
  { GrKey_Alt_N               , "GrKey_Alt_N" },
  { GrKey_Alt_M               , "GrKey_Alt_M" },
  { GrKey_Alt_Comma           , "GrKey_Alt_Comma" },
  { GrKey_Alt_Period          , "GrKey_Alt_Period" },
  { GrKey_Alt_Slash           , "GrKey_Alt_Slash" },
  { GrKey_Alt_KPStar          , "GrKey_Alt_KPStar" },
  { GrKey_F1                  , "GrKey_F1" },
  { GrKey_F2                  , "GrKey_F2" },
  { GrKey_F3                  , "GrKey_F3" },
  { GrKey_F4                  , "GrKey_F4" },
  { GrKey_F5                  , "GrKey_F5" },
  { GrKey_F6                  , "GrKey_F6" },
  { GrKey_F7                  , "GrKey_F7" },
  { GrKey_F8                  , "GrKey_F8" },
  { GrKey_F9                  , "GrKey_F9" },
  { GrKey_F10                 , "GrKey_F10" },
  { GrKey_Home                , "GrKey_Home" },
  { GrKey_Up                  , "GrKey_Up" },
  { GrKey_PageUp              , "GrKey_PageUp" },
  { GrKey_Alt_KPMinus         , "GrKey_Alt_KPMinus" },
  { GrKey_Left                , "GrKey_Left" },
  { GrKey_Center              , "GrKey_Center" },
  { GrKey_Right               , "GrKey_Right" },
  { GrKey_Alt_KPPlus          , "GrKey_Alt_KPPlus" },
  { GrKey_End                 , "GrKey_End" },
  { GrKey_Down                , "GrKey_Down" },
  { GrKey_PageDown            , "GrKey_PageDown" },
  { GrKey_Insert              , "GrKey_Insert" },
  { GrKey_Delete              , "GrKey_Delete" },
  { GrKey_Shift_F1            , "GrKey_Shift_F1" },
  { GrKey_Shift_F2            , "GrKey_Shift_F2" },
  { GrKey_Shift_F3            , "GrKey_Shift_F3" },
  { GrKey_Shift_F4            , "GrKey_Shift_F4" },
  { GrKey_Shift_F5            , "GrKey_Shift_F5" },
  { GrKey_Shift_F6            , "GrKey_Shift_F6" },
  { GrKey_Shift_F7            , "GrKey_Shift_F7" },
  { GrKey_Shift_F8            , "GrKey_Shift_F8" },
  { GrKey_Shift_F9            , "GrKey_Shift_F9" },
  { GrKey_Shift_F10           , "GrKey_Shift_F10" },
  { GrKey_Control_F1          , "GrKey_Control_F1" },
  { GrKey_Control_F2          , "GrKey_Control_F2" },
  { GrKey_Control_F3          , "GrKey_Control_F3" },
  { GrKey_Control_F4          , "GrKey_Control_F4" },
  { GrKey_Control_F5          , "GrKey_Control_F5" },
  { GrKey_Control_F6          , "GrKey_Control_F6" },
  { GrKey_Control_F7          , "GrKey_Control_F7" },
  { GrKey_Control_F8          , "GrKey_Control_F8" },
  { GrKey_Control_F9          , "GrKey_Control_F9" },
  { GrKey_Control_F10         , "GrKey_Control_F10" },
  { GrKey_Alt_F1              , "GrKey_Alt_F1" },
  { GrKey_Alt_F2            , "GrKey_Alt_F2" },
  { GrKey_Alt_F3            , "GrKey_Alt_F3" },
  { GrKey_Alt_F4            , "GrKey_Alt_F4" },
  { GrKey_Alt_F5            , "GrKey_Alt_F5" },
  { GrKey_Alt_F6            , "GrKey_Alt_F6" },
  { GrKey_Alt_F7            , "GrKey_Alt_F7" },
  { GrKey_Alt_F8            , "GrKey_Alt_F8" },
  { GrKey_Alt_F9            , "GrKey_Alt_F9" },
  { GrKey_Alt_F10           , "GrKey_Alt_F10" },
  { GrKey_Control_Print     , "GrKey_Control_Print" },
  { GrKey_Control_Left      , "GrKey_Control_Left" },
  { GrKey_Control_Right     , "GrKey_Control_Right" },
  { GrKey_Control_End       , "GrKey_Control_End" },
  { GrKey_Control_PageDown  , "GrKey_Control_PageDown" },
  { GrKey_Control_Home      , "GrKey_Control_Home" },
  { GrKey_Alt_1             , "GrKey_Alt_1" },
  { GrKey_Alt_2             , "GrKey_Alt_2" },
  { GrKey_Alt_3             , "GrKey_Alt_3" },
  { GrKey_Alt_4             , "GrKey_Alt_4" },
  { GrKey_Alt_5             , "GrKey_Alt_5" },
  { GrKey_Alt_6             , "GrKey_Alt_6" },
  { GrKey_Alt_7             , "GrKey_Alt_7" },
  { GrKey_Alt_8             , "GrKey_Alt_8" },
  { GrKey_Alt_9             , "GrKey_Alt_9" },
  { GrKey_Alt_0             , "GrKey_Alt_0" },
  { GrKey_Alt_Dash          , "GrKey_Alt_Dash" },
  { GrKey_Alt_Equals        , "GrKey_Alt_Equals" },
  { GrKey_Control_PageUp    , "GrKey_Control_PageUp" },
  { GrKey_F11               , "GrKey_F11" },
  { GrKey_F12               , "GrKey_F12" },
  { GrKey_Shift_F11         , "GrKey_Shift_F11" },
  { GrKey_Shift_F12         , "GrKey_Shift_F12" },
  { GrKey_Control_F11       , "GrKey_Control_F11" },
  { GrKey_Control_F12       , "GrKey_Control_F12" },
  { GrKey_Alt_F11           , "GrKey_Alt_F11" },
  { GrKey_Alt_F12           , "GrKey_Alt_F12" },
  { GrKey_Control_Up        , "GrKey_Control_Up" },
  { GrKey_Control_KPDash    , "GrKey_Control_KPDash" },
  { GrKey_Control_Center    , "GrKey_Control_Center" },
  { GrKey_Control_KPPlus    , "GrKey_Control_KPPlus" },
  { GrKey_Control_Down      , "GrKey_Control_Down" },
  { GrKey_Control_Insert    , "GrKey_Control_Insert" },
  { GrKey_Control_Delete    , "GrKey_Control_Delete" },
  { GrKey_Control_Tab       , "GrKey_Control_Tab" },
  { GrKey_Control_KPSlash   , "GrKey_Control_KPSlash" },
  { GrKey_Control_KPStar    , "GrKey_Control_KPStar" },
  { GrKey_Alt_KPSlash       , "GrKey_Alt_KPSlash" },
  { GrKey_Alt_Tab            , "GrKey_Alt_Tab" },
  { GrKey_Alt_Enter          , "GrKey_Alt_Enter" },
  { GrKey_Alt_LAngle         , "GrKey_Alt_LAngle" },
  { GrKey_Alt_RAngle         , "GrKey_Alt_RAngle" },
  { GrKey_Alt_At             , "GrKey_Alt_At" },
  { GrKey_Alt_LBrace         , "GrKey_Alt_LBrace" },
  { GrKey_Alt_Pipe           , "GrKey_Alt_Pipe" },
  { GrKey_Alt_RBrace         , "GrKey_Alt_RBrace" },
  { GrKey_Print              , "GrKey_Print" },
  { GrKey_Shift_Insert       , "GrKey_Shift_Insert" },
  { GrKey_Shift_Home         , "GrKey_Shift_Home" },
  { GrKey_Shift_End          , "GrKey_Shift_End" },
  { GrKey_Shift_PageUp       , "GrKey_Shift_PageUp" },
  { GrKey_Shift_PageDown     , "GrKey_Shift_PageDown" },
  { GrKey_Alt_Up             , "GrKey_Alt_Up" },
  { GrKey_Alt_Left           , "GrKey_Alt_Left" },
  { GrKey_Alt_Center         , "GrKey_Alt_Center" },
  { GrKey_Alt_Right          , "GrKey_Alt_Right" },
  { GrKey_Alt_Down           , "GrKey_Alt_Down" },
  { GrKey_Alt_Insert         , "GrKey_Alt_Insert" },
  { GrKey_Alt_Delete         , "GrKey_Alt_Delete" },
  { GrKey_Alt_Home           , "GrKey_Alt_Home" },
  { GrKey_Alt_End            , "GrKey_Alt_End" },
  { GrKey_Alt_PageUp         , "GrKey_Alt_PageUp" },
  { GrKey_Alt_PageDown       , "GrKey_Alt_PageDown" },
  { GrKey_Shift_Up           , "GrKey_Shift_Up" },
  { GrKey_Shift_Down         , "GrKey_Shift_Down" },
  { GrKey_Shift_Right        , "GrKey_Shift_Right" },
  { GrKey_Shift_Left         , "GrKey_Shift_Left" }
};

#define KEYS (sizeof(Keys)/sizeof(Keys[0]))

static GrTextOption opt;
static int curx = 0, cury = 0;

static void gputc(int c)
{
if(c == '\n' || curx + GrCharWidth(c, &opt) > GrSizeX()) {
    cury += GrCharHeight('A', &opt);
    curx = 0;
    if(cury + GrCharHeight('A', &opt) > GrSizeY()) {
GrClearScreen(opt.txo_bgcolor.v);
cury = 0;
    }
}
```

```
if(c != '\n') {
    GrDrawChar(c, curx, cury, &opt);
    curx += GrCharWidth(c, &opt);
}
}

static void gprintf(const char *format, ...)
{
va_list ap;
char buffer[0x100], *s;

va_start(ap, format);
vsprintf(buffer, format, ap);
va_end(ap);

for(s = buffer; *s; s++) gputc(*s);
}

TESTFUNC(keys) {
  int spaces_count = 0;
  KeyEntry *kp;
  GrKeyType k;
  int ok;

  opt.txo_font = &GrFont_PC8x16;
  opt.txo_fgcolor.v = GrWhite();
  opt.txo_bgcolor.v = GrBlack();
  opt.txo_chrtype = GR_BYTE_TEXT;
  opt.txo_direct = GR_TEXT_RIGHT;
  opt.txo_xalign = GR_ALIGN_LEFT;
  opt.txo_yalign = GR_ALIGN_TOP;

  gprintf("\n\n Checking GrKey... style interface"
   "\n Type 3 spaces to quit the test\n\n");
  while (spaces_count < 3) {
#ifdef PENTIUM_CLOCK
    int keyPressed=0;
    do
    {
static double old_tm = -1.0;
double  tm;
unsigned s,e;
s = rdtsc();
keyPressed = GrKeyPressed();
e = rdtsc();
tm = ((double) (e-s))/(1000.0*PENTIUM_CLOCK);
if (fabs(tm-old_tm) > 0.01) {
  gprintf ("%5.2f ",tm);
  fflush (stdout);
  old_tm = tm;
}
    } while (!keyPressed);
#endif /* PENTIUM_CLOCK */
    k = GrKeyRead();
    if (k == ’ ’) ++spaces_count; else spaces_count = 0;

    ok = 0;
    for ( kp = Keys; kp < &Keys[KEYS]; ++kp ) {
      if (k == kp->key) {
gprintf("code 0x%04x    symbol %s\n", (unsigned)k, kp->name);
ok = 1;
break;
      }
    }
    if (!ok)
      gprintf("code 0x%04x    symbol UNKNOWN\n", (unsigned)k);
  }

  gprintf("\n\n Now checking getch()"
   "\n Type 3 spaces to quit the test\n\n");
  spaces_count = 0;
  while (spaces_count < 3) {
    k = getch();
    if (k == ’ ’) ++spaces_count; else spaces_count = 0;

    gprintf("code 0x%02x      char ", (unsigned)k);
    if (ISPRINT(k))
      gprintf("’%c’\n", (char)k);
    else
      gprintf("not printable\n");

  }

  gprintf("\n\n Now checking getkey()"
   "\n Type 3 spaces to quit the test\n\n");
  spaces_count = 0;
  while (spaces_count < 3) {
    k = getkey();
    if (k == ’ ’) ++spaces_count; else spaces_count = 0;

    gprintf("code 0x%04x    char ", (unsigned)k);
    if (ISPRINT(k))
      gprintf("’%c’\n", (char)k);
    else
```

```
          gprintf("not printable\n");
      }
    }
```

life.c

```
/**
 ** life.c ---- Conway's life program
 **
 ** Copyright (c) 1995 Csaba Biegl, 820 Stirrup Dr, Nashville, TN 37221
 ** [e-mail: csaba@vuse.vanderbilt.edu]
 **
 ** This is a test/demo file of the GRX graphics library.
 ** You can use GRX test/demo files as you want.
 **
 ** The GRX graphics library is free software; you can redistribute it
 ** and/or modify it under some conditions; see the "copying.grx" file
 ** for details.
 **
 ** This library is distributed in the hope that it will be useful,
 ** but WITHOUT ANY WARRANTY; without even the implied warranty of
 ** MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 **
 **/

#include "test.h"  /* see [test.h], page 150 */
#include "rand.h"  /* see [rand.h], page 149 */

#include <malloc.h>
#include <string.h>
#include <time.h>

TESTFUNC(life)
{
int  W = GrSizeX();
int  H = GrSizeY();
char **map[2],**old,**cur;
int  *xp,*xn,*yp,*yn;
int  which,x,y,gen;
GrColor c[2];
long thresh;
for(which = 0; which < 2; which++) {
    cur = malloc(H * sizeof(char *));
    if(!cur) return;
    map[which] = cur;
    for(y = 0; y < H; y++) {
cur[y] = malloc(W);
```

```
if(!cur[y]) return;
    }
}
xp = malloc(W * sizeof(int));
xn = malloc(W * sizeof(int));
yp = malloc(H * sizeof(int));
yn = malloc(H * sizeof(int));
if(!xp || !xn || !yp || !yn) return;
for(x = 0; x < W; x++) {
    xp[x] = (x + W - 1) % W;
    xn[x] = (x + W + 1) % W;
}
for(y = 0; y < H; y++) {
    yp[y] = (y + H - 1) % H;
    yn[y] = (y + H + 1) % H;
}
c[0] = GrBlack();
c[1] = GrWhite();
which = 0;
old = map[which];
cur = map[1 - which];
        SRND((int)time(NULL));
for(y = 0; y < H; y++) {
    for(x = 0; x < W; x++) {
int ii = RND() % 53;
while(--ii >= 0) RND();
old[y][x] = (((RND() % 131) > 107) ? 1 : 0);
GrPlotNC(x,y,c[(int)old[y][x]]);
    }
}
thresh = (((unsigned long)RND() << 16) + RND()) % 1003567UL;
gen    = (Argc > 0) ? 1 : 0;
do {
    for(y = 0; y < H; y++) {
char *prow = old[yp[y]];
char *crow = old[y];
char *nrow = old[yn[y]];
char *curr = cur[y];
for(x = 0; x < W; x++) {
    int  xprev = xp[x];
    int  xnext = xn[x];
    char live  = prow[xprev] +
 prow[x]     +
 prow[xnext] +
 crow[xprev] +
 crow[xnext] +
 nrow[xprev] +
 nrow[x]     +
 nrow[xnext];
    live = ((live | crow[x]) == 3) ? 1 : 0;
    if(--thresh <= 0) {
live  ^= gen;
thresh = (((unsigned long)RND() << 16) + RND()) % 1483567UL;
    }
    curr[x] = live;
}
    }
    for(y = 0; y < H; y++) {
char *curr = cur[y];
char *oldr = old[y];
for(x = 0; x < W; x++) {
    if(curr[x] != oldr[x]) GrPlotNC(x,y,c[(int)curr[x]]);
}
    }
    which = 1 - which;
    old = map[which];
    cur = map[1 - which];
} while(!GrKeyPressed());
while(GrKeyPressed()) GrKeyRead();
}
```

linetest.c

```
/**
 ** linetest.c ---- test wide and patterned lines
 **
 ** Copyright (c) 1995 Csaba Biegl, 820 Stirrup Dr, Nashville, TN 37221
 ** [e-mail: csaba@vuse.vanderbilt.edu]
 **
 ** This is a test/demo file of the GRX graphics library.
 ** You can use GRX test/demo files as you want.
 **
 ** The GRX graphics library is free software; you can redistribute it
 ** and/or modify it under some conditions; see the "copying.grx" file
 ** for details.
 **
 ** This library is distributed in the hope that it will be useful,
 ** but WITHOUT ANY WARRANTY; without even the implied warranty of
 ** MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 **
 **/

#include "test.h"  /* see [test.h], page 150 */
```

```
TESTFUNC(test1)
{
GrLineOption o1,o2,o3,o4;
int i;
for(i = 0; i < 2; i++) {
    o1.lno_color   = GrAllocColor(255,0,0);
    o1.lno_width   = 1;
    o1.lno_pattlen = 4 * i;
    o1.lno_dashpat = "\5\5\24\24";
    o2.lno_color   = GrAllocColor(255,255,0);
    o2.lno_width   = 2;
    o2.lno_pattlen = 6 * i;
    o2.lno_dashpat = "\5\5\24\24\2\2";
    o3.lno_color   = GrAllocColor(0,255,255);
    o3.lno_width   = 30;
    o3.lno_pattlen = 8 * i;
    o3.lno_dashpat = "\5\5\24\24\2\2\40\40";
    o4.lno_color   = GrAllocColor(255,0,255);
    o4.lno_width   = 4;
    o4.lno_pattlen = 6 * i;
    o4.lno_dashpat = "\2\2\2\2\10\10";
    GrClearScreen(GrBlack());
    GrCustomLine(10,10,100,100,&o1);
    GrCustomLine(10,50,100,140,&o1);
    GrCustomLine(10,90,100,180,&o1);
    GrCustomLine(110,10,200,100,&o2);
    GrCustomLine(110,50,200,140,&o2);
    GrCustomLine(110,90,200,180,&o2);
    GrCustomLine(210,10,300,100,&o3);
    GrCustomLine(210,50,300,140,&o3);
    GrCustomLine(210,90,300,180,&o3);
    GrCustomLine(20,300,600,300,&o4);
    GrCustomLine(20,320,600,340,&o4);
    GrCustomLine(20,380,600,360,&o4);
    GrCustomLine(400,100,400,300,&o4);
    GrCustomLine(420,100,440,300,&o4);
    GrCustomLine(480,100,460,300,&o4);
    GrCustomLine(600,200,500,300,&o4);
    GrKeyRead();
    GrClearScreen(GrBlack());
    GrCustomBox(50,50,550,350,&o3);
    GrCustomCircle(300,200,50,&o2);
    GrKeyRead();
}
}
```

memtest.c

```
/**
 ** memtest.c ---- test memory driver
 **
 ** Copyright (C) 2001 Mariano Alvarez Fernandez
 ** [e-mail: malfer@teleline.es]
 **
 ** This is a test/demo file of the GRX graphics library.
 ** You can use GRX test/demo files as you want.
 **
 ** The GRX graphics library is free software; you can redistribute it
 ** and/or modify it under some conditions; see the "copying.grx" file
 ** for details.
 **
 ** This library is distributed in the hope that it will be useful,
 ** but WITHOUT ANY WARRANTY; without even the implied warranty of
 ** MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 **
 **/

#include <stdio.h>
#include "grx20.h"
#include "drawing.h"  /* see [drawing.h], page 146 */

int main()
{
  int x, y, ww, wh;
  GrColor c;
  GrContext *w1, *w2, *w3, *w4;

  GrSetDriver( "memory gw 400 gh 400 nc 256" );
  GrSetMode( GR_default_graphics );

  x = GrSizeX();
  y = GrSizeY();
  ww = (x / 2) - 10;
  wh = (y / 2) - 10;
  w1 = GrCreateSubContext(5,5,ww+4,wh+4,NULL,NULL);
  w2 = GrCreateSubContext(15+ww,5,ww+ww+14,wh+4,NULL,NULL);
  w3 = GrCreateSubContext(5,15+wh,ww+4,wh+wh+14,NULL,NULL);
  w4 = GrCreateSubContext(15+ww,15+wh,ww+ww+14,wh+wh+14,NULL,NULL);

  GrSetContext(w1);
  c = GrAllocColor(200,100,100);
  drawing(0,0,ww,wh,c,GrBlack());
  GrBox(0,0,ww-1,wh-1,c);
  GrSetContext(w2);
  c = GrAllocColor(100,200,200);
  drawing(0,0,ww,wh,c,GrBlack());
  GrBox(0,0,ww-1,wh-1,c);

  GrSetContext(w3);
  c = GrAllocColor(200,200,0);
  drawing(0,0,ww,wh,c,GrBlack());
  GrBox(0,0,ww-1,wh-1,c);

  GrSetContext(w4);
  c = GrAllocColor(0,100,200);
  drawing(0,0,ww,wh,c,GrBlack());
  GrBox(0,0,ww-1,wh-1,c);

  GrSetContext( NULL );
//  GrSaveBmpImage( "memtest.bmp",NULL,0,0,639,479 );
  GrSaveContextToPpm( NULL,"memtest.ppm","GRX MemTest" );

  return 0;
}
```

mousetst.c

```
/**
 ** mousetst.c ---- test mouse cursor and mouse/keyboard input
 **
 ** Copyright (c) 1995 Csaba Biegl, 820 Stirrup Dr, Nashville, TN 37221
 ** [e-mail: csaba@vuse.vanderbilt.edu]
 **
 ** This is a test/demo file of the GRX graphics library.
 ** You can use GRX test/demo files as you want.
 **
 ** The GRX graphics library is free software; you can redistribute it
 ** and/or modify it under some conditions; see the "copying.grx" file
 ** for details.
 **
 ** This library is distributed in the hope that it will be useful,
 ** but WITHOUT ANY WARRANTY; without even the implied warranty of
 ** MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 **
 **/

#include <string.h>
#include <stdio.h>
#include <ctype.h>
```

```
#include "test.h"  /* see [test.h], page 150 */

TESTFUNC(mousetest)
{
GrMouseEvent evt;
GrColor bgc = GrAllocColor(0,0,128);
GrColor fgc = GrAllocColor(255,255,0);
int  testmotion = 0;
int  ii,mode;

if(GrMouseDetect()) {
    GrMouseEventMode(1);
    GrMouseInit();
    GrMouseSetColors(GrAllocColor(255,0,0),GrBlack());
    GrMouseDisplayCursor();
    GrClearScreen(bgc);
    ii = 0;
    mode = GR_M_CUR_NORMAL;
    GrTextXY(
10,(GrScreenY() - 20),
"Commands: 'N' -- next mouse mode, 'Q' -- exit",
GrWhite(),
bgc
    );
    for( ; ; ) {
char msg[200];
drawing(ii,ii,(GrSizeX() - 20),(GrSizeY() - 20),((fgc ^ bgc) | GrXOR),GrNOCOLOR);
GrMouseGetEventT(GR_M_EVENT,&evt,0L);
if(evt.flags & (GR_M_KEYPRESS | GR_M_BUTTON_CHANGE | testmotion)) {
    strcpy(msg,"Got event(s): ");
#                   define mend (&msg[strlen(msg)])
    if(evt.flags & GR_M_MOTION)      strcpy( mend,"[moved] ");
    if(evt.flags & GR_M_LEFT_DOWN)   strcpy( mend,"[left down] ");
    if(evt.flags & GR_M_MIDDLE_DOWN) strcpy( mend,"[middle down] ");
    if(evt.flags & GR_M_RIGHT_DOWN)  strcpy( mend,"[right down] ");
    if(evt.flags & GR_M_LEFT_UP)     strcpy( mend,"[left up] ");
    if(evt.flags & GR_M_MIDDLE_UP)   strcpy( mend,"[middle up] ");
    if(evt.flags & GR_M_RIGHT_UP)    strcpy( mend,"[right up] ");
    if(evt.flags & GR_M_KEYPRESS)    sprintf(mend,"[key (0x%03x)] ",evt.key);
    sprintf(mend,"at X=%d, Y=%d, ",evt.x,evt.y);
    sprintf(mend,
"buttons=%c%c%c, ",
(evt.buttons & GR_M_LEFT)   ? 'L' : 'l',
(evt.buttons & GR_M_MIDDLE) ? 'M' : 'm',
(evt.buttons & GR_M_RIGHT)  ? 'R' : 'r'
    );
    sprintf(mend,"deltaT=%ld (ms)",evt.dtime);
    strcpy (mend,"                          ");
    GrTextXY(10,(GrScreenY() - 40),msg,GrWhite(),bgc);
    testmotion = evt.buttons ? GR_M_MOTION : 0;
}
if(evt.flags & GR_M_KEYPRESS) {
    int key = evt.key;
    if((key == ’Q’) || (key == ’q’)) break;
    if((key != ’N’) && (key != ’n’)) continue;
    GrMouseEraseCursor();
    switch(mode = (mode + 1) & 3) {
      case GR_M_CUR_RUBBER:
GrMouseSetCursorMode(GR_M_CUR_RUBBER,evt.x,evt.y,GrWhite() ^ bgc);
break;
      case GR_M_CUR_LINE:
GrMouseSetCursorMode(GR_M_CUR_LINE,evt.x,evt.y,GrWhite() ^ bgc);
break;
      case GR_M_CUR_BOX:
GrMouseSetCursorMode(GR_M_CUR_BOX,-20,-10,20,10,GrWhite() ^ bgc);
break;
      default:
GrMouseSetCursorMode(GR_M_CUR_NORMAL);
break;
    }
    GrMouseDisplayCursor();
}
if((ii += 7) > 20) ii -= 20;
    }
    GrMouseUnInit();
} else {
    GrClearScreen(bgc);
    ii = 0;
    mode = GR_M_CUR_NORMAL;
    GrTextXY(
(GrScreenX()/3),(GrScreenY() - 20),
"Sorry, no mouse found !",
GrWhite(),
bgc
    );
}
}
```

pcirctst.c

```
/**
 ** pcirctst.c ---- test custom circle and ellipse rendering
 **
 ** Copyright (c) 1995 Csaba Biegl, 820 Stirrup Dr, Nashville, TN 37221
 ** [e-mail: csaba@vuse.vanderbilt.edu]
 **
 ** This is a test/demo file of the GRX graphics library.
 ** You can use GRX test/demo files as you want.
 **
 ** The GRX graphics library is free software; you can redistribute it
 ** and/or modify it under some conditions; see the "copying.grx" file
 ** for details.
 **
 ** This library is distributed in the hope that it will be useful,
 ** but WITHOUT ANY WARRANTY; without even the implied warranty of
 ** MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 **
 **/

#include "test.h"  /* see [test.h], page 150 */
#include <math.h>

static int stop = 0;

static int widths[] = { 1, 2, 5, 10, 20, 50, 0 };

static GrLineOption Solid = { 0, 1, 0, NULL };  /* normal solid */

static GrLineOption *Patterns[] = {
  &Solid, NULL
};

void drawellip(int xc,int yc,int xa,int ya,GrColor c1,GrColor c2,GrColor c3)
{
double ddx = (double)xa;
double ddy = (double)ya;
double R2 = ddx*ddx*ddy*ddy;
double SQ;
int x1,x2,y1,y2;
int dx,dy;
int *wdt, idx;
GrLineOption *l;

for (idx = 0, l = *Patterns; l != NULL; l = Patterns[++idx])
    for (wdt=widths; *wdt != 0; ++wdt) {
GrClearScreen(GrBlack());

GrFilledBox(xc-xa,yc-ya,xc+xa,yc+ya,c1);
dx = xa;
dy = 0;
GrPlot(xc-dx,yc,c3);
GrPlot(xc+dx,yc,c3);
while(++dy <= ya) {
    SQ = R2 - (double)dy * (double)dy * ddx * ddx;
    dx = (int)(sqrt(SQ)/ddy + 0.5);
    x1 = xc - dx;
    x2 = xc + dx;
    y1 = yc - dy;
    y2 = yc + dy;
    GrPlot(x1,y1,c3);
    GrPlot(x2,y1,c3);
    GrPlot(x1,y2,c3);
    GrPlot(x2,y2,c3);
}

l->lno_color = c2;
l->lno_width = *wdt;
GrCustomEllipse(xc,yc,xa,ya,l);
if(GrKeyRead() == 'q') {
  stop = 1;
  return;
}
    }
}

TESTFUNC(circtest)
{
int  xc,yc;
int  xr,yr;
GrColor c1,c2,c3;

c1 = GrAllocColor(64,64,255);
c2 = GrAllocColor(255,255,64);
c3 = GrAllocColor(255,64,64);
xc = GrSizeX() / 2;
yc = GrSizeY() / 2;
xr = 1;
yr = 1;
while(!stop && ((xr < 1000) || (yr < 1000))) {
    drawellip(xc,yc,xr,yr,c1,c2,c3);
    xr += xr/4+1;
    yr += yr/4+1;
}
xr = 4;
yr = 1;
while(!stop && ((xr < 1000) || (yr < 1000))) {
    drawellip(xc,yc,xr,yr,c1,c2,c3);
    yr += yr/4+1;
```

```
        xr = yr * 4;
    }
    xr = 1;
    yr = 4;
    while(!stop && ((xr < 1000) || (yr < 1000))) {
        drawellip(xc,yc,xr,yr,c1,c2,c3);
        xr += xr/4+1;
        yr = xr * 4;
    }
    }
```

pngtest.c

```
/**
 ** pngtest.c ---- test the ctx2png routines
 **
 ** Copyright (c) 2001 Mariano Alvarez Fernandez
 ** [e-mail: malfer@teleline.es]
 **
 ** This is a test/demo file of the GRX graphics library.
 ** You can use GRX test/demo files as you want.
 **
 ** The GRX graphics library is free software; you can redistribute it
 ** and/or modify it under some conditions; see the "copying.grx" file
 ** for details.
 **
 ** This library is distributed in the hope that it will be useful,
 ** but WITHOUT ANY WARRANTY; without even the implied warranty of
 ** MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 **/

#include <stdio.h>
#include <stdlib.h>
#include "grx20.h"
#include "grxkeys.h"

void imagen( char *nf )
{
  GrContext *grc;
  int wide, high;
  char s[81];
  int w, h;

  GrQueryPng( nf,&w,&h );
  sprintf( s,"%s %dx%d",nf,w,h );
  wide = (w > 300) ? 300 : w;
  high = (h > 400) ? 400 : h;
  GrClearScreen( GrAllocColor( 0,0,200 ) );

  GrBox( 10,40,10+wide+1,40+high+1,GrWhite() );
  grc = GrCreateSubContext( 11,41,11+wide-1,41+high-1,NULL,NULL );
  GrLoadContextFromPng( grc,nf,0 );
  GrDestroyContext( grc );

  GrBox( 320,40,320+wide+1,40+high+1,GrWhite() );
  grc = GrCreateSubContext( 321,41,321+wide-1,41+high-1,NULL,NULL );
  GrLoadContextFromPng( grc,nf,1 );
  GrDestroyContext( grc );

  GrTextXY( 10,10,s,GrBlack(),GrWhite() );
  GrTextXY( 10,50+high,"Press any key to continue",GrBlack(),GrWhite() );
  GrKeyRead();
}

void nopngsupport( void )
{
  char *s[6] = {
    "Warning!",
    "You need libpng (http://www.libpng.org/pub/png/libpng.html)",
    "and enable png support in the GRX lib (edit makedefs.grx)",
    "to run this demo",
    " ",
    "Press any key to continue..." };
  int i;

  GrClearScreen( GrAllocColor( 0,0,100 ) );
  for( i=0; i<6; i++ )
    GrTextXY( 90,160+i*18,s[i],GrWhite(),GrNOCOLOR );
  GrKeyRead();
}


int main()
{
  GrContext *grc;

  GrSetMode( GR_width_height_bpp_graphics,640,480,24 );

  if( !GrPngSupport() ){
    nopngsupport();
    GrSetMode(GR_default_text);
    exit( 1 );
    }
  imagen( "pngcompo.png" );
  imagen( "pngowl.png" );
  imagen( "pngred.png" );

  GrClearScreen( GrAllocColor( 0,100,0 ) );
  grc = GrCreateSubContext( 191,121,191+256-1,121+240-1,NULL,NULL );
  GrLoadContextFromPng( grc,"pngred.png",1 );
  GrDestroyContext( grc );
  grc = GrCreateSubContext( 181,241,181+289-1,241+80-1,NULL,NULL );
  GrLoadContextFromPng( grc,"pngcompo.png",1 );
  GrDestroyContext( grc );

  GrTextXY( 10,10,"Press any key to save screen",GrBlack(),GrWhite() );
  GrKeyRead();
  GrSaveContextToPng( NULL,"output.png" );

  GrClearScreen( GrBlack() );
  GrTextXY( 10,10,"Press any key to reload screen",GrBlack(),GrWhite() );
  GrKeyRead();
  GrLoadContextFromPng( NULL,"output.png",0 );

  GrTextXY( 10,10,"Press any key to end          ",GrBlack(),GrWhite() );
  GrKeyRead();
  GrSetMode(GR_default_text);
  return 0;
}
```

pnmtest.c

```
/**
 ** pnmtest.c ---- test the ctx2pnm routines
 **
 ** Copyright (c) 2000 Mariano Alvarez Fernandez
 ** [e-mail: malfer@teleline.es]
 **
 ** This is a test/demo file of the GRX graphics library.
 ** You can use GRX test/demo files as you want.
 **
 ** The GRX graphics library is free software; you can redistribute it
 ** and/or modify it under some conditions; see the "copying.grx" file
 ** for details.
 **
 ** This library is distributed in the hope that it will be useful,
 ** but WITHOUT ANY WARRANTY; without even the implied warranty of
 ** MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 **/
```

```
#include <stdlib.h>
#include <stdio.h>
#include "grx20.h"
#include "grxkeys.h"

#define FIMAGEPPM "pnmtest.ppm"
#define FIMAGEPBM "pnmtest.pbm"

#define FIMAGEPGM "prueba.pgm"
#define FIMAGEPBM2 "prueba.pbm"
#define FSCREEN "output.ppm"

int main(void)
{
  GrContext *grc;
  int wide, high, maxval;
  char s[81];

/* GrSetMode( GR_default_graphics ); */
  GrSetMode( GR_width_height_color_graphics,640,480,32768 );
  GrQueryPnm( FIMAGEPPM, &wide, &high, &maxval );
  sprintf( s,"%s %d x %d pixels",FIMAGEPPM,wide,high );
  GrTextXY( 10,20,s,GrBlack(),GrWhite() );
  GrBox( 10,40,10+wide+1,40+high+1,GrWhite() );
  grc = GrCreateSubContext( 11,41,11+wide-1,41+high-1,NULL,NULL );
  GrLoadContextFromPnm( grc,FIMAGEPPM );
  GrSaveContextToPgm( grc,FIMAGEPGM,"TestPnm" );
  GrDestroyContext( grc );
  GrTextXY( 10,50+high,"Press RETURN to continue",GrBlack(),GrWhite() );
  GrKeyRead();

  GrClearScreen( GrBlack() );
  GrQueryPnm( FIMAGEPGM, &wide, &high, &maxval );
  sprintf( s,"%s %d x %d pixels",FIMAGEPGM,wide,high );
  GrTextXY( 10,20,s,GrBlack(),GrWhite() );
  GrBox( 10,40,10+wide+1,40+high+1,GrWhite() );
  grc = GrCreateSubContext( 11,41,11+wide-1,41+high-1,NULL,NULL );
  GrLoadContextFromPnm( grc,FIMAGEPGM );
  GrDestroyContext( grc );
  GrTextXY( 10,50+high,"Press RETURN to continue",GrBlack(),GrWhite() );
  GrKeyRead();

  GrClearScreen( GrBlack() );
  GrQueryPnm( FIMAGEPBM, &wide, &high, &maxval );
  sprintf( s,"%s %d x %d pixels",FIMAGEPBM,wide,high );
  GrTextXY( 10,20,s,GrBlack(),GrWhite() );
  GrBox( 10,40,10+wide+1,40+high+1,GrWhite() );
  grc = GrCreateSubContext( 11,41,11+wide-1,41+high-1,NULL,NULL );
  GrLoadContextFromPnm( grc,FIMAGEPBM );
  GrSaveContextToPbm( grc,FIMAGEPBM2,"TestPnm" );
  GrDestroyContext( grc );
  GrTextXY( 10,50+high,"Press RETURN to continue",GrBlack(),GrWhite() );
  GrKeyRead();

  GrClearScreen( GrBlack() );
  GrQueryPnm( FIMAGEPPM, &wide, &high, &maxval );
  GrBox( 10,40,10+wide+1,40+high+1,GrWhite() );
  grc = GrCreateSubContext( 11,41,11+wide-1,41+high-1,NULL,NULL );
  GrLoadContextFromPnm( grc,FIMAGEPPM );
  GrDestroyContext( grc );
  GrQueryPnm( FIMAGEPGM, &wide, &high, &maxval );
  GrBox( 110,140,110+wide+1,140+high+1,GrWhite() );
  grc = GrCreateSubContext( 111,141,111+wide-1,141+high-1,NULL,NULL );
  GrLoadContextFromPnm( grc,FIMAGEPGM );
  GrDestroyContext( grc );
  GrQueryPnm( FIMAGEPBM, &wide, &high, &maxval );
  GrBox( 210,240,210+wide+1,240+high+1,GrWhite() );
  grc = GrCreateSubContext( 211,241,211+wide-1,241+high-1,NULL,NULL );
  GrLoadContextFromPnm( grc,FIMAGEPBM2 );
  GrDestroyContext( grc );
  GrTextXY( 10,20,"Press RETURN to save screen",GrBlack(),GrWhite() );
  GrKeyRead();

  GrSaveContextToPpm( NULL,FSCREEN,"TestPnm" );
  GrClearScreen( GrWhite() );
  GrTextXY( 10,20,"Press RETURN to reload screen",GrWhite(),GrBlack() );
  GrKeyRead();

  GrLoadContextFromPnm( NULL,FSCREEN );
  GrTextXY( 10,20,"Press RETURN to end        ",GrBlack(),GrWhite() );
  GrKeyRead();

  GrSetMode(GR_default_text);
  return 0;
}
```

polytest.c

```
/**
 ** polytest.c ---- test polygon rendering
 **
 ** Copyright (c) 1995 Csaba Biegl, 820 Stirrup Dr, Nashville, TN 37221
 ** [e-mail: csaba@vuse.vanderbilt.edu]
 **
 ** This is a test/demo file of the GRX graphics library.
 ** You can use GRX test/demo files as you want.
 **
 ** The GRX graphics library is free software; you can redistribute it
 ** and/or modify it under some conditions; see the "copying.grx" file
 ** for details.
 **
 ** This library is distributed in the hope that it will be useful,
 ** but WITHOUT ANY WARRANTY; without even the implied warranty of
 ** MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 **
 **/

#include <string.h>
#include <stdio.h>
#include <time.h>

#ifndef  CLK_TCK
#define  CLK_TCK    CLOCKS_PER_SEC
#endif

#include "test.h"  /* see [test.h], page 150 */

static GrColor *EGA;
#define black EGA[0]
#define red   EGA[12]
#define blue  EGA[1]
#define white EGA[15]

static void testpoly(int n,int points[][2],int convex)
{
GrClearScreen(black);
GrPolygon(n,points,white);
GrFilledPolygon(n,points,(red | GrXOR));
GrKeyRead();
if(convex || (n <= 3)) {
    GrClearScreen(black);
    GrFilledPolygon(n,points,white);
    GrFilledConvexPolygon(n,points,(red | GrXOR));
    GrKeyRead();
}
}

static void speedtest(void)
{
int pts[4][2];
int ww = GrSizeX() / 10;
int hh = GrSizeY() / 10;
int sx = (GrSizeX() - 2*ww) / 32;
int sy = (GrSizeY() - 2*hh) / 32;
int  ii,jj;
GrColor color;
long t1,t2,t3;

GrClearScreen(black);
t1 = clock();
pts[0][1] = 0;
pts[1][1] = hh;
pts[2][1] = 2*hh;
pts[3][1] = hh;
color = 0;
for(ii = 0; ii < 32; ii++) {
    pts[0][0] = ww;
    pts[1][0] = 2*ww;
    pts[2][0] = ww;
    pts[3][0] = 0;
    for(jj = 0; jj < 32; jj++) {
GrFilledPolygon(4,pts, EGA[color] | GrXOR);
color = (color + 1) & 15;
pts[0][0] += sx;
pts[1][0] += sx;
pts[2][0] += sx;
pts[3][0] += sx;
    }
    pts[0][1] += sy;
    pts[1][1] += sy;
    pts[2][1] += sy;
    pts[3][1] += sy;
}
t2 = clock();
pts[0][1] = 0;
pts[1][1] = hh;
pts[2][1] = 2*hh;
pts[3][1] = hh;
color = 0;
for(ii = 0; ii < 32; ii++) {
    pts[0][0] = ww;
    pts[1][0] = 2*ww;
    pts[2][0] = ww;
    pts[3][0] = 0;
    for(jj = 0; jj < 32; jj++) {
GrFilledConvexPolygon(4,pts, EGA[color] | GrXOR);
color = (color + 1) & 15;
pts[0][0] += sx;
pts[1][0] += sx;
pts[2][0] += sx;
pts[3][0] += sx;
    }
    pts[0][1] += sy;
    pts[1][1] += sy;
    pts[2][1] += sy;
    pts[3][1] += sy;
}
t3 = clock();
sprintf(exit_message,
    "Times to scan 1024 polygons\n"
    "   with 'GrFilledPolygon': %.2f (s)\n"
    "   with 'GrFilledConvexPolygon': %.2f (s)\n",
    (double)(t2 - t1) / (double)CLK_TCK,
    (double)(t3 - t2) / (double)CLK_TCK
);
}

TESTFUNC(ptest)
{
char buff[300];
int  pts[300][2];
int  ii,collect;
int  convex;
FILE *fp;

fp = fopen("polytest.dat","r");  /* see [polytest.dat], page 152 */
if(fp == NULL) return;
EGA = GrAllocEgaColors();
ii  = collect = convex = 0;
while(fgets(buff,299,fp) != NULL) {
    if(!collect) {
if(strncmp(buff,"begin",5) == 0) {
    convex  = (buff[5] == 'c');
    collect = 1;
    ii      = 0;
}
continue;
    }
    if(strncmp(buff,"end",3) == 0) {
testpoly(ii,pts,convex);
collect = 0;
continue;
    }
    if(sscanf(buff,"%d %d",&pts[ii][0],&pts[ii][1]) == 2) ii++;
```

```
}
fclose(fp);
speedtest();
}
```

rgbtest.c

```
/**
 ** rgbtest.c ---- show 256 color RGB palette
 **
 ** Copyright (c) 1995 Csaba Biegl, 820 Stirrup Dr, Nashville, TN 37221
 ** [e-mail: csaba@vuse.vanderbilt.edu]
 **
 ** This is a test/demo file of the GRX graphics library.
 ** You can use GRX test/demo files as you want.
 **
 ** The GRX graphics library is free software; you can redistribute it
 ** and/or modify it under some conditions; see the "copying.grx" file
 ** for details.
 **
 ** This library is distributed in the hope that it will be useful,
 ** but WITHOUT ANY WARRANTY; without even the implied warranty of
 ** MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 **
 **/

#include "test.h"  /* see [test.h], page 150 */

TESTFUNC(rgbtest)
{
int x = GrSizeX();
int y = GrSizeY();
int ww = (x-10)/32;
int wh = (y-10)/8;
int ii,jj;

GrSetRGBcolorMode();
for(ii = 0; ii < 8; ii++) {
    for(jj = 0; jj < 32; jj++) {
GrFilledBox(5+jj*ww,5+ii*wh,5+jj*ww+ww-1,5+ii*wh+wh-1,ii*32+jj);
    }
}
GrKeyRead();
}
```

sbctest.c

```
/**
 ** sbctest.c ---- test subcontext
 **
 ** This is a test/demo file of the GRX graphics library.
 ** You can use GRX test/demo files as you want.
 **
 ** The GRX graphics library is free software; you can redistribute it
 ** and/or modify it under some conditions; see the "copying.grx" file
 ** for details.
 **
 ** This library is distributed in the hope that it will be useful,
 ** but WITHOUT ANY WARRANTY; without even the implied warranty of
 ** MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 **
 **/

#include <string.h>
#include "test.h"  /* see [test.h], page 150 */

static void drawpf( int border, GrPattern *pat );
static void drawp( int border, GrLinePattern *grlp );

TESTFUNC(sbctest)
{
  char bits[] = {0, 76, 50, 0, 0, 76, 60, 0};
  GrContext *grc, *grsc;
  GrPattern *pat1, *pat2;
  GrLineOption grl;
  GrLinePattern grlp;
  GrFont *grf;
  GrTextOption grt;

  grc = GrCreateContext( 300,300,NULL,NULL );
  if( grc == NULL ) return;
  grsc = GrCreateSubContext( 10,10,290,290,grc,NULL );
  if( grsc == NULL ) return;
  pat1 = GrBuildPixmapFromBits( bits,8,8,GrWhite(),GrBlack() );
  if( pat1 == NULL ) return;
  pat2 = GrBuildPixmapFromBits( bits,8,8,GrBlack(),GrWhite() );
  if( pat2 == NULL ) return;
  grf = GrLoadFont( "lucb40.fnt" );
  if( grf == NULL ){
    grf = GrLoadFont( "../fonts/lucb40.fnt" );
    if( grf == NULL ) return;
    }
```

```
GrBox( 19,19,320,320,GrWhite() );

GrTextXY( 0,0,"White drawing on context       ",GrWhite(),GrBlack() );
GrSetContext( grc );
GrClearContext( GrBlack() );
drawing( 10,10,280,280,GrWhite(),GrNOCOLOR );
GrSetContext( NULL );
GrBitBlt( NULL,20,20,grc,0,0,299,299,GrWRITE );
GrKeyRead();

GrTextXY( 0,0,"Black drawing on subcontext    ",GrWhite(),GrBlack() );
GrSetContext( grsc );
drawing( 0,0,280,280,GrBlack(),GrNOCOLOR );
GrSetContext( NULL );
GrBitBlt( NULL,20,20,grc,0,0,299,299,GrWRITE );
GrKeyRead();

GrTextXY( 0,0,"Pattern drawing on context     ",GrWhite(),GrBlack() );
GrSetContext( grc );
GrClearContext( GrBlack() );
drawpf( 10,pat1 );
GrSetContext( NULL );
GrBitBlt( NULL,20,20,grc,0,0,299,299,GrWRITE );
GrKeyRead();

GrTextXY( 0,0,"Pattern drawing on subcontext  ",GrWhite(),GrBlack() );
GrSetContext( grsc );
GrClearContext( GrBlack() );
drawpf( 0,pat2 );
GrSetContext( NULL );
GrBitBlt( NULL,20,20,grc,0,0,299,299,GrXOR );
GrKeyRead();

grl.lno_color = GrWhite();
grl.lno_width = 3;
grl.lno_pattlen = 0;
grlp.lnp_pattern = pat1;
grlp.lnp_option = &grl;

GrTextXY( 0,0,"Patterned drawing on context   ",GrWhite(),GrBlack() );
GrSetContext( grc );
GrClearContext( GrBlack() );
grlp.lnp_pattern = pat1;
drawp( 10,&grlp );
GrSetContext( NULL );
GrBitBlt( NULL,20,20,grc,0,0,299,299,GrWRITE );
```

```
  GrKeyRead();

  GrTextXY( 0,0,"Patterned drawing on subcontext",GrWhite(),GrBlack() );
  GrSetContext( grsc );
  GrClearContext( GrBlack() );
  grlp.lnp_pattern = pat2;
  drawp( 0,&grlp );
  GrSetContext( NULL );
  GrBitBlt( NULL,20,20,grc,0,0,299,299,GrXOR );
  GrKeyRead();

  grt.txo_fgcolor.v = GrWhite();
  grt.txo_bgcolor.v = GrBlack() | GrOR;
  grt.txo_font = grf;
  grt.txo_direct = GR_TEXT_RIGHT;
  grt.txo_xalign = GR_ALIGN_LEFT;
  grt.txo_yalign = GR_ALIGN_CENTER;
  grt.txo_chrtype = GR_BYTE_TEXT;

  GrTextXY( 0,0,"Patterned text on context      ",GrWhite(),GrBlack() );
  GrSetContext( grc );
  GrClearContext( GrBlack() );
  GrPatternDrawString( "Hello all",9,20,60,&grt,pat1 );
  GrPatternDrawChar( 'G',20,120,&grt,pat1 );
  GrPatternDrawStringExt( "Hola a todos",12,20,180,&grt,pat1 );
  GrSetContext( NULL );
  GrBitBlt( NULL,20,20,grc,0,0,299,299,GrWRITE );
  GrKeyRead();

  GrTextXY( 0,0,"Patterned text on subcontext   ",GrWhite(),GrBlack() );
  GrSetContext( grsc );
  GrClearContext( GrBlack() );
  GrPatternDrawString( "Hello all",9,10,50,&grt,pat2 );
  GrPatternDrawChar( 'G',10,110,&grt,pat2 );
  GrPatternDrawStringExt( "Hola a todos",12,10,170,&grt,pat2 );
  GrSetContext( NULL );
  GrBitBlt( NULL,20,20,grc,0,0,299,299,GrXOR );
  GrKeyRead();

  GrUnloadFont( grf );
  GrDestroyPattern( pat2 );
  GrDestroyPattern( pat1 );
  GrDestroyContext( grsc );
  GrDestroyContext( grc );
}

/***/
```

```
static void drawpf( int border, GrPattern *pat )
{
  int pt1[4][2] = {{130,200},{140,240},{150,250},{160,180}};
  int pt2[4][2] = {{230,200},{235,240},{246,250},{258,180}};
  int ptaux[4][2];
  int i,j;

  GrPatternFilledBox( 0+border,0+border,93+border,93+border,pat );
  GrPatternFilledCircle( 139+border,46+border,45,pat );
  GrPatternFilledEllipse( 232+border,46+border,45,35,pat );
  GrPatternFilledCircleArc( 46+border,139+border,45,-300,600,
                            GR_ARC_STYLE_CLOSE2,pat );
  GrPatternFilledEllipseArc( 139+border,139+border,45,35,-700,400,
                             GR_ARC_STYLE_CLOSE2,pat );
  GrPatternFilledLine( 188+border,139+border,278+border,139+border,pat );
  GrPatternFilledPlot( 47+border,228+border,pat );
  GrPatternFilledPlot( 47+border,229+border,pat );
  GrPatternFilledPlot( 47+border,230+border,pat );
  GrPatternFilledPlot( 47+border,231+border,pat );
  GrPatternFilledPlot( 47+border,232+border,pat );
  for( i=0; i<4; i++ )
    for( j=0; j<2; j++ )
      ptaux[i][j] = pt1[i][j] + border;
  GrPatternFilledPolygon( 4,ptaux,pat );
  for( i=0; i<4; i++ )
    for( j=0; j<2; j++ )
      ptaux[i][j] = pt2[i][j] + border;
  GrPatternFilledConvexPolygon( 4,ptaux,pat );
}

/***/

static void drawp( int border, GrLinePattern *grlp )
{
  int pt1[4][2] = {{130,200},{140,240},{150,250},{160,180}};
  int pt2[4][2] = {{230,200},{235,240},{246,250},{258,180}};
  int ptaux[4][2];
  int i,j;

  GrPatternedBox( 0+border,0+border,93+border,93+border,grlp );
  GrPatternedCircle( 139+border,46+border,45,grlp );
  GrPatternedEllipse( 232+border,46+border,45,35,grlp );
  GrPatternedCircleArc( 46+border,139+border,45,-300,600,
                        GR_ARC_STYLE_CLOSE2,grlp );
  GrPatternedEllipseArc( 139+border,139+border,45,35,-700,400,
                         GR_ARC_STYLE_CLOSE2,grlp );
  GrPatternedLine( 188+border,139+border,278+border,139+border,grlp );
  for( i=0; i<4; i++ )
    for( j=0; j<2; j++ )
      ptaux[i][j] = pt1[i][j] + border;
  GrPatternedPolygon( 4,ptaux,grlp );
  for( i=0; i<4; i++ )
    for( j=0; j<2; j++ )
      ptaux[i][j] = pt2[i][j] + border;
  GrPatternedPolyLine( 4,ptaux,grlp );
}
```

scroltst.c

```
/**
 ** scroltst.c ---- test virtual screen set/scroll
 **
 ** Copyright (c) 1995 Csaba Biegl, 820 Stirrup Dr, Nashville, TN 37221
 ** [e-mail: csaba@vuse.vanderbilt.edu]
 **
 ** This is a test/demo file of the GRX graphics library.
 ** You can use GRX test/demo files as you want.
 **
 ** The GRX graphics library is free software; you can redistribute it
 ** and/or modify it under some conditions; see the "copying.grx" file
 ** for details.
 **
 ** This library is distributed in the hope that it will be useful,
 ** but WITHOUT ANY WARRANTY; without even the implied warranty of
 ** MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 **
 **/

#include "test.h"  /* see [test.h], page 150 */

TESTFUNC(scrolltest)
{
int  wdt = GrScreenX();
int  hgt = GrScreenY();
GrColor nc  = GrNumColors();
int  txh = GrDefaultFont.h.height + 2;
for( ; ; ) {
    char buff[100];
    char *l1 = "Screen resolution: %dx%d";
    char *l2 = "Virtual resolution: %dx%d";
    char *l3 = "Current screen start: x=%-4d y=%-4d";
    char *l4 = "Commands: q -- exit program,";
    char *l5 = "w W h H -- shrink/GROW screen width or height,";
    char *l6 = "x X y Y -- decrease/INCREASE screen start position";
    GrColor bgc = GrAllocColor(0,0,128);
    GrColor fgc = GrAllocColor(200,200,0);
    GrColor txc = GrAllocColor(255,0,255);
    int vw = GrVirtualX();
    int vh = GrVirtualY();
    int vx = GrViewportX();
    int vy = GrViewportY();
    int x  = (vw / 2) - (strlen(l6) * GrDefaultFont.h.width / 2);
    int y  = (vh / 2) - (3 * txh);
    GrClearScreen(bgc);
    drawing(0,0,vw,vh,fgc,bgc);
    sprintf(buff,l1,wdt,hgt); GrTextXY(x,y,buff,txc,bgc); y += txh;
    sprintf(buff,l2,vw, vh ); GrTextXY(x,y,buff,txc,bgc); y += txh;
    for( ; ; GrSetViewport(vx,vy)) {
int yy = y;
vx = GrViewportX();
vy = GrViewportY();
sprintf(buff,l3,vx,vy); GrTextXY(x,yy,buff,txc,bgc); yy += txh;
GrTextXY(x,yy,l4,txc,bgc); yy += txh;
GrTextXY(x,yy,l5,txc,bgc); yy += txh;
GrTextXY(x,yy,l6,txc,bgc); yy += txh;
switch(GrKeyRead()) {
    case 'w': vw -= 8; break;
    case 'W': vw += 8; break;
    case 'h': vh -= 8; break;
    case 'H': vh += 8; break;
    case 'x': vx--; continue;
    case 'X': vx++; continue;
    case 'y': vy--; continue;
    case 'Y': vy++; continue;
    case 'q': return;
    case 'Q': return;
    default:  continue;
}
GrSetMode(GR_custom_graphics,wdt,hgt,nc,vw,vh);
break;
    }
}
}
```

speedtst.c

```
/**
 ** speedtst.c ---- check all available frame drivers speed
 **
 ** Copyright (c) 1995 Csaba Biegl, 820 Stirrup Dr, Nashville, TN 37221
 ** [e-mail: csaba@vuse.vanderbilt.edu]
 **
 ** This is a test/demo file of the GRX graphics library.
 ** You can use GRX test/demo files as you want.
 **
 ** The GRX graphics library is free software; you can redistribute it
 ** and/or modify it under some conditions; see the "copying.grx" file
 ** for details.
 **
 ** This library is distributed in the hope that it will be useful,
 ** but WITHOUT ANY WARRANTY; without even the implied warranty of
 ** MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 **
 **/

#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <ctype.h>
#include <assert.h>
#ifdef __WATCOMC__
/*#include <wcdefs.h>*/
#include <conio.h>
#else
#include <values.h>
#endif
#include <math.h>
#include <time.h>

#include "rand.h"  /* see [rand.h], page 149 */

#include "grx20.h"

#if GRX_VERSION_API-0 <= 0x0220
#define GrColor unsigned long
#define BLIT_FAIL(gp) ((gp)->fm!=GR_frameVGA8X)
#else
#define BLIT_FAIL(gp)  0
#endif

#define MEASURE_RAM_MODES 1

#define READPIX_loops      (384*1)
#define READPIX_X11_loops  (  4*1)
#define DRAWPIX_loops      (256*1)
```

```
#define DRAWLIN_loops      (12*1)
#define DRAWHLIN_loops     (16*1)
#define DRAWVLIN_loops     (12*1)
#define DRAWBLK_loops      (1*1)
#define BLIT_loops         (1*1)

typedef struct {
    double rate, count;
} perfm;

typedef struct {
    GrFrameMode fm;
    int    w,h,bpp;
    int    flags;
    perfm  readpix;
    perfm  drawpix;
    perfm  drawlin;
    perfm  drawhlin;
    perfm  drawvlin;
    perfm  drawblk;
    perfm  blitv2v;
    perfm  blitv2r;
    perfm  blitr2v;
} gvmode;
#define FLG_measured 0x0001
#define FLG_tagged   0x0002
#define FLG_rammode  0x0004
#define MEASURED(g) (((g)->flags&FLG_measured)!=0)
#define TAGGED(g)   (((g)->flags&FLG_tagged)!=0)
#define RAMMODE(g)  (((g)->flags&FLG_rammode)!=0)
#define SET_MEASURED(g)  (g)->flags |= FLG_measured
#define SET_TAGGED(g)    (g)->flags |= FLG_tagged
#define SET_RAMMODE(g)   (g)->flags |= FLG_rammode
#define TOGGLE_TAGGED(g) (g)->flags ^= FLG_tagged

int  nmodes = 0;
#define MAX_MODES 256
gvmode *grmodes = NULL;
#if MEASURE_RAM_MODES
gvmode *rammodes = NULL;
#endif

/* No of Points [(x,y) pairs]. Must be multiple of 2*3=6 */
#define PAIRS 4200

#define UL(x)  ((unsigned long)(x))
#define DBL(x)  ((double)(x))
```

```
#define INT(x) ((int)(x))

#ifndef  CLK_TCK
#define  CLK_TCK    CLOCKS_PER_SEC
#endif

#ifndef min
#define min(a,b) ((a)<(b) ? (a) : (b))
#endif
#ifndef max
#define max(a,b) ((a)>(b) ? (a) : (b))
#endif

typedef struct XYpairs {
  int x[PAIRS];
  int y[PAIRS];
  int w, h;
  struct XYpairs *nxt;
} XY_PAIRS;

XY_PAIRS *xyp = NULL;
int *xb = NULL, *yb = NULL; /* need sorted pairs for block operations */
int measured_any = 0;

XY_PAIRS *checkpairs(int w, int h) {
  XY_PAIRS *res = xyp;
  int i;

  if (xb == NULL) {
    xb = malloc(sizeof(int) * PAIRS);
    yb = malloc(sizeof(int) * PAIRS);
  }

  while (res != NULL) {
    if (res->w == w && res->h == h)
      return res;
    res = res->nxt;
  }

  SRND(12345);

  res = malloc(sizeof(XY_PAIRS));
  assert(res != NULL);
  res->w = w;
  res->h = h;
  res->nxt = xyp;
  xyp = res;
  for (i=0; i < PAIRS; ++i) {
    int x = RND() % w;
    int y = RND() % h;
    if (x < 0) x = 0; else
    if (x >=w) x = w-1;
    if (y < 0) y = 0; else
    if (y >=h) y = h-1;
    res->x[i] = x;
    res->y[i] = y;
  }
  return res;
}

double SQR(int a, int b) {
  double r = DBL(a-b);
  return r*r;
}

double ABS(int a, int b) {
  double r = DBL(a-b);
  return fabs(r);
}

char *FrameDriverName(GrFrameMode m) {

#if GRX_VERSION_API-0 >= 0x0229
  unsigned sys = GrGetLibrarySystem();
#else
  unsigned sys = (unsigned) GRX_VERSION;
#endif

  int x11 = ( (sys == GRX_VERSION_GENERIC_X11) ||
              (sys == GRX_VERSION_GCC_386_X11) ||
              (sys == GRX_VERSION_GCC_X86_64_X11) );
  int w32 = ( (sys == GRX_VERSION_GCC_386_WIN32) ||
              (sys == GRX_VERSION_MSC_386_WIN32) ||
      (sys == GRX_VERSION_GCC_386_CYG32) );
  int sdl = strcmp( GrCurrentVideoDriver()->name , "sdl") == 0;

  switch(m) {
    case GR_frameUndef: return "Undef";
    case GR_frameText : return "Text";
    case GR_frameHERC1: return "HERC1";
    case GR_frameEGAVGA1: return x11 ? "XWIN1" : w32 ? "WIN32_1" : "EGAVGA1";
    case GR_frameEGA4: return "EGA4";
    case GR_frameSVGA4: return x11 ? "XWIN4" : w32 ? "WIN32_4" : "SVGA4";
    case GR_frameSVGA8: return sdl ? "SDL8" : x11 ? "XWIN8" : w32 ? "WIN32_8" : "SVGA8
    case GR_frameVGA8X: return "VGA8X";
    case GR_frameSVGA16: return sdl ? "SDL16" : x11 ? "XWIN16" : w32 ? "WIN32_16" : "S
    case GR_frameSVGA24: return sdl ? "SDL24" : x11 ? "XWIN24" : w32 ? "WIN32_24" : "S
    case GR_frameSVGA32L: return sdl ? "SDL32L" : x11 ? "XWIN32L" : w32 ? "WIN32_32L"
    case GR_frameSVGA32H: return sdl ? "SDL32H" : x11 ? "XWIN32H" : w32 ? "WIN32_32H"
    case GR_frameSVGA8_LFB: return "LFB8";
    case GR_frameSVGA16_LFB: return "LFB16";
    case GR_frameSVGA24_LFB: return "LFB24";
    case GR_frameSVGA32L_LFB: return "LFB32L";
    case GR_frameSVGA32H_LFB: return "LFB32H";
    case GR_frameRAM1: return "RAM1";
    case GR_frameRAM4: return "RAM4";
    case GR_frameRAM8: return "RAM8";
    case GR_frameRAM16: return "RAM16";
    case GR_frameRAM24: return "RAM24";
    case GR_frameRAM32L: return "RAM32L";
    case GR_frameRAM32H: return "RAM32H";
    case GR_frameRAM3x8: return "RAM3x8";
    default: return "UNKNOWN";
  }
  return "UNKNOWN";
}

void Message(int disp, char *txt, gvmode *gp) {
  char msg[200];
  sprintf(msg, "%s: %d x %d x %dbpp",
FrameDriverName(gp->fm), gp->w, gp->h, gp->bpp);
#if GRX_VERSION_API-0 >= 0x0229
  unsigned sys = GrGetLibrarySystem();
#else
  unsigned sys = (unsigned) GRX_VERSION;
#endif
  if ( (sys == GRX_VERSION_GENERIC_X11) ||
       (sys == GRX_VERSION_GCC_386_X11) ||
       (sys == GRX_VERSION_GCC_X86_64_X11) )
    fprintf(stderr,"%s\t%s\n", msg, txt);
  if (disp) {
    GrTextOption to;
    GrContext save;
    GrSaveContext(&save);
    GrSetContext(NULL);
    to.txo_font = &GrFont_PC6x8;
    to.txo_fgcolor.v = GrWhite();
    to.txo_bgcolor.v = GrBlack();
    to.txo_chrtype = GR_BYTE_TEXT;
    to.txo_direct  = GR_TEXT_RIGHT;
    to.txo_xalign  = GR_ALIGN_LEFT;
    to.txo_yalign  = GR_ALIGN_TOP;
    GrDrawString(msg,strlen(msg),0,0,&to);
    GrDrawString(txt,strlen(txt),0,10,&to);
    GrSetContext(&save);
  }
}

void printresultheader(FILE *f) {
  fprintf(f, "Driver               readp drawp line   hline vline  block  v2v    v2r
}

void printresultline(FILE *f, gvmode * gp) {
  fprintf(f, "%-9s %4dx%4d ", FrameDriverName(gp->fm), gp->w, gp->h);
  fprintf(f, "%6.2f", gp->readpix.rate  / (1024.0 * 1024.0));
  fprintf(f, "%6.2f", gp->drawpix.rate  / (1024.0 * 1024.0));
  fprintf(f, "%6.2f", gp->drawlin.rate  / (1024.0 * 1024.0));
  fprintf(f, "%7.2f", gp->drawhlin.rate / (1024.0 * 1024.0));
  fprintf(f, "%6.2f", gp->drawvlin.rate / (1024.0 * 1024.0));
  fprintf(f, "%7.2f", gp->drawblk.rate  / (1024.0 * 1024.0));
  fprintf(f, "%7.2f", gp->blitv2v.rate  / (1024.0 * 1024.0));
  fprintf(f, "%7.2f", gp->blitv2r.rate  / (1024.0 * 1024.0));
  fprintf(f, "%7.2f", gp->blitr2v.rate  / (1024.0 * 1024.0));
  fprintf(f, "\n");
}

void readpixeltest(gvmode *gp, XY_PAIRS *pairs,int loops) {
  int i, j;
  long t1,t2;
  double seconds;
  int *x = pairs->x;
  int *y = pairs->y;

  if (!MEASURED(gp)) {
    gp->readpix.rate  = 0.0;
    gp->readpix.count = DBL(PAIRS) * DBL(loops);
  }

  t1 = clock();
  for (i=loops; i > 0; --i) {
    for (j=PAIRS-1; j >= 0; j--)
       GrPixelNC(x[j],y[j]);
  }
  t2 = clock();
  seconds = DBL(t2 - t1) / DBL(CLK_TCK);
  if (seconds > 0)
    gp->readpix.rate = gp->readpix.count / seconds;
}
```

```
void drawpixeltest(gvmode *gp, XY_PAIRS *pairs) {
  int i, j;
  GrColor c1 = GrWhite();
  GrColor c2 = GrWhite() | GrXOR;
  GrColor c3 = GrWhite() | GrOR;
  GrColor c4 = GrBlack() | GrAND;
  long t1,t2;
  double seconds;
  int *x = pairs->x;
  int *y = pairs->y;

  if (!MEASURED(gp)) {
    gp->drawpix.rate  = 0.0;
    gp->drawpix.count = DBL(PAIRS) * DBL(DRAWPIX_loops) * 4.0;
  }

  t1 = clock();
  for (i=0; i < DRAWPIX_loops; ++i) {
    for (j=PAIRS-1; j >= 0; j--) GrPlotNC(x[j],y[j],c1);
    for (j=PAIRS-1; j >= 0; j--) GrPlotNC(x[j],y[j],c2);
    for (j=PAIRS-1; j >= 0; j--) GrPlotNC(x[j],y[j],c3);
    for (j=PAIRS-1; j >= 0; j--) GrPlotNC(x[j],y[j],c4);
  }
  t2 = clock();
  seconds = DBL(t2 - t1) / DBL(CLK_TCK);
  if (seconds > 0)
    gp->drawpix.rate = gp->drawpix.count / seconds;
}

void drawlinetest(gvmode *gp, XY_PAIRS *pairs) {
  int i, j;
  int *x = pairs->x;
  int *y = pairs->y;
  GrColor c1 = GrWhite();
  GrColor c2 = GrWhite() | GrXOR;
  GrColor c3 = GrWhite() | GrOR;
  GrColor c4 = GrBlack() | GrAND;
  long t1,t2;
  double seconds;

  if (!MEASURED(gp)) {
    gp->drawlin.rate  = 0.0;
    gp->drawlin.count = 0.0;
    for (j=0; j < PAIRS; j+=2)
      gp->drawlin.count += sqrt(SQR(x[j],x[j+1])+SQR(y[j],y[j+1]));
    gp->drawlin.count *= 4.0 * DRAWLIN_loops;
  }

  t1 = clock();
  for (i=0; i < DRAWLIN_loops; ++i) {
    for (j=PAIRS-2; j >= 0; j-=2)
GrLineNC(x[j],y[j],x[j+1],y[j+1],c1);
    for (j=PAIRS-2; j >= 0; j-=2)
GrLineNC(x[j],y[j],x[j+1],y[j+1],c2);
    for (j=PAIRS-2; j >= 0; j-=2)
GrLineNC(x[j],y[j],x[j+1],y[j+1],c3);
    for (j=PAIRS-2; j >= 0; j-=2)
GrLineNC(x[j],y[j],x[j+1],y[j+1],c4);
  }
  t2 = clock();
  seconds = DBL(t2 - t1) / DBL(CLK_TCK);
  if (seconds > 0)
    gp->drawlin.rate = gp->drawlin.count / seconds;
}

void drawhlinetest(gvmode *gp, XY_PAIRS *pairs) {
  int  i, j;
  int *x = pairs->x;
  int *y = pairs->y;
  GrColor c1 = GrWhite();
  GrColor c2 = GrWhite() | GrXOR;
  GrColor c3 = GrWhite() | GrOR;
  GrColor c4 = GrBlack() | GrAND;
  long t1,t2;
  double seconds;

  if (!MEASURED(gp)) {
    gp->drawhlin.rate = 0.0;
    gp->drawhlin.count = 0.0;
    for (j=0; j < PAIRS; j+=2)
      gp->drawhlin.count += ABS(x[j],x[j+1]);
    gp->drawhlin.count *= 4.0 * DRAWHLIN_loops;
  }

  t1 = clock();
  for (i=0; i < DRAWHLIN_loops; ++i) {
    for (j=PAIRS-2; j >= 0; j-=2)
      GrHLineNC(x[j],x[j+1],y[j],c1);
    for (j=PAIRS-2; j >= 0; j-=2)
      GrHLineNC(x[j],x[j+1],y[j],c2);
    for (j=PAIRS-2; j >= 0; j-=2)
      GrHLineNC(x[j],x[j+1],y[j],c3);
    for (j=PAIRS-2; j >= 0; j-=2)
```

```
        GrHLineNC(x[j],x[j+1],y[j],c4);
    }
    t2 = clock();
    seconds = DBL(t2 - t1) / DBL(CLK_TCK);
    if (seconds > 0)
      gp->drawhlin.rate = gp->drawhlin.count / seconds;
  }

  void drawvlinetest(gvmode *gp, XY_PAIRS *pairs) {
    int i, j;
    int *x = pairs->x;
    int *y = pairs->y;
    GrColor c1 = GrWhite();
    GrColor c2 = GrWhite() | GrXOR;
    GrColor c3 = GrWhite() | GrOR;
    GrColor c4 = GrBlack() | GrAND;
    long t1,t2;
    double seconds;

    if (!MEASURED(gp)) {
      gp->drawvlin.rate = 0.0;
      gp->drawvlin.count = 0.0;
      for (j=0; j < PAIRS; j+=2)
        gp->drawvlin.count += ABS(y[j],y[j+1]);
      gp->drawvlin.count *= 4.0 * DRAWVLIN_loops;
    }

    t1 = clock();
    for (i=0; i < DRAWVLIN_loops; ++i) {
      for (j=PAIRS-2; j >= 0; j-=2)
         GrVLineNC(x[j],y[j],y[j+1],c1);
      for (j=PAIRS-2; j >= 0; j-=2)
         GrVLineNC(x[j],y[j],y[j+1],c2);
      for (j=PAIRS-2; j >= 0; j-=2)
         GrVLineNC(x[j],y[j],y[j+1],c3);
      for (j=PAIRS-2; j >= 0; j-=2)
         GrVLineNC(x[j],y[j],y[j+1],c4);
    }
    t2 = clock();
    seconds = DBL(t2 - t1) / DBL(CLK_TCK);
    if (seconds > 0)
      gp->drawvlin.rate = gp->drawvlin.count / seconds;
  }

  void drawblocktest(gvmode *gp, XY_PAIRS *pairs) {
    int i, j;
    GrColor c1 = GrWhite();
    GrColor c2 = GrWhite() | GrXOR;
    GrColor c3 = GrWhite() | GrOR;
    GrColor c4 = GrBlack() | GrAND;
    long t1,t2;
    double seconds;

    if (xb == NULL || yb == NULL) return;

    for (j=0; j < PAIRS; j+=2) {
      xb[j]   = min(pairs->x[j],pairs->x[j+1]);
      xb[j+1] = max(pairs->x[j],pairs->x[j+1]);
      yb[j]   = min(pairs->y[j],pairs->y[j+1]);
      yb[j+1] = max(pairs->y[j],pairs->y[j+1]);
    }

    if (!MEASURED(gp)) {
      gp->drawblk.rate = 0.0;
      gp->drawblk.count = 0.0;
      for (j=0; j < PAIRS; j+=2)
        gp->drawblk.count += ABS(xb[j],xb[j+1]) * ABS(yb[j],yb[j+1]);
      gp->drawblk.count *= 4.0 * DRAWBLK_loops;
    }

    t1 = clock();
    for (i=0; i < DRAWBLK_loops; ++i) {
      for (j=PAIRS-2; j >= 0; j-=2)
        GrFilledBoxNC(xb[j],yb[j],xb[j+1],yb[j+1],c1);
      for (j=PAIRS-2; j >= 0; j-=2)
        GrFilledBoxNC(xb[j],yb[j],xb[j+1],yb[j+1],c2);
      for (j=PAIRS-2; j >= 0; j-=2)
        GrFilledBoxNC(xb[j],yb[j],xb[j+1],yb[j+1],c3);
      for (j=PAIRS-2; j >= 0; j-=2)
        GrFilledBoxNC(xb[j],yb[j],xb[j+1],yb[j+1],c4);
    }
    t2 = clock();
    seconds = DBL(t2 - t1) / DBL(CLK_TCK);
    if (seconds > 0)
      gp->drawblk.rate = gp->drawblk.count / seconds;
  }

  void xor_draw_blocks(GrContext *c) {
    GrContext save;
    int i;

    GrSaveContext(&save);
    GrSetContext(c);
    GrClearContext(GrBlack());
    for (i=28; i > 1; --i)
      GrFilledBox(GrMaxX()/i,GrMaxY()/i,
  (i-1)*GrMaxX()/i,(i-1)*GrMaxY()/i,GrWhite()|GrXOR);
    GrSetContext(&save);
  }

  void blit_measure(gvmode *gp, perfm *p,
    int *xb, int *yb,
    GrContext *dst,GrContext *src) {
    int i, j;
    long t1,t2;
    double seconds;
    GrContext save;

    GrSaveContext(&save);
    if (dst != src) {
      GrSetContext(dst);
      GrClearContext(GrBlack());
    }
    xor_draw_blocks(src);
    GrSetContext(&save);

    if (dst != NULL) {
      char *s = src != NULL ? "ram" : "video";
      char *d = dst != NULL ? "ram" : "video";
      char txt[50];
      sprintf(txt, "blit test: %s -> %s", s, d);
      Message(1,txt, gp);
    }

    t1 = clock();
    for (i=0; i < BLIT_loops; ++i) {
      for (j=PAIRS-3; j >= 0; j-=3)
        GrBitBlt(dst,xb[j+2],yb[j+2],src,xb[j+1],yb[j+1],xb[j],yb[j],GrWRITE);
      for (j=PAIRS-3; j >= 0; j-=3)
        GrBitBlt(dst,xb[j+2],yb[j+2],src,xb[j+1],yb[j+1],xb[j],yb[j],GrXOR);
      for (j=PAIRS-3; j >= 0; j-=3)
        GrBitBlt(dst,xb[j+2],yb[j+2],src,xb[j+1],yb[j+1],xb[j],yb[j],GrOR);
      for (j=PAIRS-3; j >= 0; j-=3)
        GrBitBlt(dst,xb[j+2],yb[j+2],src,xb[j+1],yb[j+1],xb[j],yb[j],GrAND);
    }
    t2 = clock();
    seconds = DBL(t2 - t1) / DBL(CLK_TCK);
    if (seconds > 0)
      p->rate = p->count / seconds;
  }
```

```
void blittest(gvmode *gp, XY_PAIRS *pairs, int ram) {
  int j;

  if (xb == NULL || yb == NULL) return;

  for (j=0; j < PAIRS; j+=3) {
    int wh;
    xb[j]   = max(pairs->x[j],pairs->x[j+1]);
    xb[j+1] = min(pairs->x[j],pairs->x[j+1]);
    xb[j+2] = pairs->x[j+2];
    wh      = xb[j]-xb[j+1];
    if (xb[j+2]+wh >= gp->w) xb[j+2] = gp->w - wh - 1;
    yb[j]   = max(pairs->y[j],pairs->y[j+1]);
    yb[j+1] = min(pairs->y[j],pairs->y[j+1]);
    yb[j+2] = pairs->y[j+2];
    wh      = yb[j]-yb[j+1];
    if (yb[j+2]+wh >= gp->h) yb[j+2] = gp->h - wh - 1;
  }

  if (!MEASURED(gp)) {
    double count = 0.0;
    for (j=0; j < PAIRS; j+=3)
      count += ABS(xb[j],xb[j+1]) * ABS(yb[j],yb[j+1]);
    gp->blitv2v.count =
    gp->blitr2v.count =
    gp->blitv2r.count = count * 4.0 * BLIT_loops;
    gp->blitv2v.rate  =
    gp->blitr2v.rate  =
    gp->blitv2r.rate  = 0.0;
  }

#if BLIT_loops-0
  blit_measure(gp, &gp->blitv2v, xb, yb,
       (GrContext *)(RAMMODE(gp) ? GrCurrentContext() : NULL),
       (GrContext *)(RAMMODE(gp) ? GrCurrentContext() : NULL));
  if (!BLIT_FAIL(gp) && !ram) {
    GrContext rc;
    GrContext *rcp = GrCreateContext(gp->w,gp->h,NULL,&rc);
    if (rcp) {
      blit_measure(gp, &gp->blitv2r, xb, yb, rcp, NULL);
      blit_measure(gp, &gp->blitr2v, xb, yb, NULL, rcp);
      GrDestroyContext(rcp);
    }
  }
#endif
}
```

```
void measure_one(gvmode *gp, int ram) {
  XY_PAIRS *pairs;

  if (MEASURED(gp)) return;
  pairs = checkpairs(gp->w, gp->h);
  GrFilledBox( 0, 0, gp->w-1, gp->h-1, GrBlack());
  Message(RAMMODE(gp),"read pixel test", gp);
  { int rd_loops = READPIX_loops;
#if GRX_VERSION_API-0 >= 0x0229
  unsigned sys = GrGetLibrarySystem();
#else
  unsigned sys = (unsigned) GRX_VERSION;
#endif
  if ( (sys == GRX_VERSION_GENERIC_X11) ||
       (sys == GRX_VERSION_GCC_386_X11) ||
       (sys == GRX_VERSION_GCC_X86_64_X11) )
      if (!RAMMODE(gp)) rd_loops = READPIX_X11_loops;
    readpixeltest(gp,pairs,rd_loops);
  }
  GrFilledBox( 0, 0, gp->w-1, gp->h-1, GrBlack());
  Message(RAMMODE(gp),"draw pixel test", gp);
  drawpixeltest(gp,pairs);
  GrFilledBox( 0, 0, gp->w-1, gp->h-1, GrBlack());
  Message(RAMMODE(gp),"draw line test ", gp);
  drawlinetest(gp,pairs);
  GrFilledBox( 0, 0, gp->w-1, gp->h-1, GrBlack());
  Message(RAMMODE(gp),"draw hline test", gp);
  drawhlinetest(gp,pairs);
  GrFilledBox( 0, 0, gp->w-1, gp->h-1, GrBlack());
  Message(RAMMODE(gp),"draw vline test", gp);
  drawvlinetest(gp,pairs);
  GrFilledBox( 0, 0, gp->w-1, gp->h-1, GrBlack());
  Message(RAMMODE(gp),"draw block test", gp);
  drawblocktest(gp,pairs);
  GrFilledBox( 0, 0, gp->w-1, gp->h-1, GrBlack());
  blittest(gp, pairs, ram);
  GrFilledBox( 0, 0, gp->w-1, gp->h-1, GrBlack());
  SET_MEASURED(gp);
  measured_any = 1;
}

#if MEASURE_RAM_MODES
int identical_measured(gvmode *tm) {
  int i;
  for (i=0; i < nmodes; ++i) {
    if (tm      != &rammodes[i]    &&
tm->fm  == rammodes[i].fm  &&
tm->w   == rammodes[i].w   &&
tm->h   == rammodes[i].h   &&
tm->bpp == rammodes[i].bpp &&
MEASURED(&rammodes[i])        ) return (1);
  }
  return 0;
}
#endif

static int first = 0;

void speedcheck(gvmode *gp, int print, int wait) {
  char m[41];
  gvmode *rp = NULL;

  if (first) {
    printf(
      "speedtest may take some time to process.\n"
      "Now press <CR> to continue..."
    );
    fflush(stdout);
    fgets(m,40,stdin);
  }

  GrSetMode(
      GR_width_height_bpp_graphics,
      gp->w, gp->h, gp->bpp
  );

  if (first) {
    /* xor_draw_blocks(NULL);
       getch(); */
    first = 0;
  }

  if ( GrScreenFrameMode() != gp->fm) {
    GrFrameMode act = GrScreenFrameMode();
    GrSetMode(GR_default_text);
    printf("Setup failed : %s != %s\n",
    FrameDriverName(act),
    FrameDriverName(gp->fm));
    fgets(m,40,stdin);
    return;
  }

  if (!MEASURED(gp))
```

```
    measure_one(gp, 0);

#if MEASURE_RAM_MODES
  rp = &rammodes[(unsigned)(gp-grmodes)];
  rp->fm = GrCoreFrameMode();
  if (!MEASURED(rp) && !identical_measured(rp)) {
    GrContext rc;
    if (GrCreateFrameContext(rp->fm,gp->w,gp->h,NULL,&rc)) {
      GrSetContext(&rc);
      measure_one(rp, 1);
      GrDestroyContext(&rc);
      GrSetContext(NULL);
    }
  }
#endif

  GrSetMode(GR_default_text);
  if (print) {
    printf("Results: \n");
    printresultheader(stdout);
    printresultline(stdout, gp);
    if (rp)
      printresultline(stdout, rp);
  }
  if (wait)
    fgets(m,40,stdin);
}

int collectmodes(const GrVideoDriver *drv)
{
gvmode *gp = grmodes;
GrFrameMode fm;
const GrVideoMode *mp;
for(fm =GR_firstGraphicsFrameMode;
      fm <= GR_lastGraphicsFrameMode; fm++) {
    for(mp = GrFirstVideoMode(fm); mp; mp = GrNextVideoMode(mp)) {
gp->fm    = fm;
gp->w     = mp->width;
gp->h     = mp->height;
gp->bpp   = mp->bpp;
gp->flags = 0;
gp++;
if (gp-grmodes >= MAX_MODES) return MAX_MODES;
    }
}
return(int)(gp-grmodes);
}
```

```
int vmcmp(const void *m1,const void *m2)
{
gvmode *md1 = (gvmode *)m1;
gvmode *md2 = (gvmode *)m2;
if(md1->bpp != md2->bpp) return(md1->bpp - md2->bpp);
if(md1->w   != md2->w  ) return(md1->w   - md2->w  );
if(md1->h   != md2->h  ) return(md1->h   - md2->h  );
return(0);
}

#define LINES   20
#define COLUMNS 80

void ModeText(int i, int shrt,char *mdtxt) {
char *flg;

if (MEASURED(&grmodes[i])) flg = " #"; else
if (TAGGED(&grmodes[i]))   flg = " *"; else
   flg = ") ";
switch (shrt) {
  case 2 : sprintf(mdtxt,"%2d%s %dx%d ", i+1, flg, grmodes[i].w, grmodes[i].h);
   break;
  case 1 : sprintf(mdtxt,"%2d%s %4dx%-4d ", i+1, flg, grmodes[i].w, grmodes[i].h);
   break;
  default: sprintf(mdtxt,"  %2d%s  %4dx%-4d ", i+1, flg, grmodes[i].w, grmodes[i].h);
   break;
}
mdtxt += strlen(mdtxt);

if (grmodes[i].bpp > 20)
  sprintf(mdtxt, "%ldM", 1L << (grmodes[i].bpp-20));
else  if (grmodes[i].bpp > 10)
  sprintf(mdtxt, "%ldk", 1L << (grmodes[i].bpp-10));
else
  sprintf(mdtxt, "%ld", 1L << grmodes[i].bpp);
switch (shrt) {
  case 2 : break;
  case 1 : strcat(mdtxt, " col"); break;
  default: strcat(mdtxt, " colors"); break;
}
}

int ColsCheck(int cols, int ml, int sep) {
  int len;

  len = ml * cols + (cols-1) * sep + 1;
  return len <= COLUMNS;
}

void PrintModes(void) {
char mdtxt[100];
unsigned int maxlen;
int i, n, shrt, c, cols;

cols = (nmodes+LINES-1) / LINES;
do {
  for (shrt = 0; shrt <= 2; ++shrt) {
    maxlen = 0;
    for (i = 0; i < nmodes; ++i) {
      ModeText(i,shrt,mdtxt);
      if (strlen(mdtxt) > maxlen) maxlen = strlen(mdtxt);
    }
    n = 2;
    if (cols>1 || shrt<2) {
      if (!ColsCheck(cols, maxlen, n)) continue;
      while (ColsCheck(cols, maxlen, n+1) && n < 4) ++n;
    }
    c = 0;
    for (i = 0; i < nmodes; ++i) {
      if (++c == cols) c = 0;
      ModeText(i,shrt,mdtxt);
      printf("%*s%s", (c ? -((int)(maxlen+n)) : -((int)maxlen)), mdtxt, (c || (i+1==nm
    }
    return;
  }
  --cols;
} while (1);
}

int main(int argc, char **argv)
{
int  i;

grmodes = malloc(MAX_MODES*sizeof(gvmode));
assert(grmodes!=NULL);
#if MEASURE_RAM_MODES
rammodes = malloc(MAX_MODES*sizeof(gvmode));
assert(rammodes!=NULL);
#endif

GrSetDriver(NULL);
if(GrCurrentVideoDriver() == NULL) {
    printf("No graphics driver found\n");
    exit(1);
}

nmodes = collectmodes(GrCurrentVideoDriver());
if(nmodes == 0) {
    printf("No graphics modes found\n");
    exit(1);
}
qsort(grmodes,nmodes,sizeof(grmodes[0]),vmcmp);
#if MEASURE_RAM_MODES
for (i=0; i < nmodes; ++i) {
  rammodes[i].fm    = GR_frameUndef;      /* filled in later */
  rammodes[i].w     = grmodes[i].w;
  rammodes[i].h     = grmodes[i].h;
  rammodes[i].bpp   = grmodes[i].bpp;
  rammodes[i].flags = FLG_rammode;
}
#endif

if(argc >= 2 && (i = atoi(argv[1])) >= 1 && i <= nmodes) {
    speedcheck(&grmodes[i - 1], 1, 0);
    goto done;
}

first = 1;
for( ; ; ) {
    char mb[41], *m = mb;
    int tflag = 0;
    GrSetMode(GR_default_text);
    printf(
"Graphics driver: \"%s\"\t"
"graphics defaults: %dx%d %ld colors\n",
GrCurrentVideoDriver()->name,
GrDriverInfo->defgw,
GrDriverInfo->defgh,
(long)GrDriverInfo->defgc
    );
    PrintModes();
    printf("\nEnter #, ’t#’ toggels tag, ’m’ measure tagged and ’q’ to quit> ");
    fflush(stdout);
    if(!fgets(m,40,stdin)) continue;
    switch (*m) {
      case ’t’:
      case ’T’: tflag = 1;
++m;
break;
      case ’A’:
```

```
      case 'a': for (i=0; i < nmodes; ++i)
  SET_TAGGED(&grmodes[i]);
break;
      case 'M':
      case 'm': for (i=0; i < nmodes; ++i)
  if (TAGGED(&grmodes[i])) {
    speedcheck(&grmodes[i], 0, 0);
    TOGGLE_TAGGED(&grmodes[i]);
  }
break;
      case 'Q':
      case 'q': goto done;
    }
    if ((sscanf(m,"%d",&i) != 1) || (i < 1) || (i > nmodes))
continue;
    i--;
    if (tflag) TOGGLE_TAGGED(&grmodes[i]);
  else speedcheck(&grmodes[i], 1, 1);
}
done:
if (measured_any) {
    int i;
    FILE *log = fopen("speedtst.log", "a");

    if (!log) exit(1);

    fprintf( log, "\nGraphics driver: \"%s\"\n\n",
       GrCurrentVideoDriver()->name);
    printf("Results: \n");
    printresultheader(log);

    for (i=0; i < nmodes; ++i)
      if (MEASURED(&grmodes[i]))
printresultline(log, &grmodes[i]);
#if MEASURE_RAM_MODES
    for (i=0; i < nmodes; ++i)
      if (MEASURED(&rammodes[i]))
printresultline(log, &rammodes[i]);
#endif
    fclose(log);
}
return(0);
}
```

textpatt.c

```
/**
 ** textpatt.c
 **
 ** This is a test/demo file of the GRX graphics library.
 ** You can use GRX test/demo files as you want.
 **
 ** The GRX graphics library is free software; you can redistribute it
 ** and/or modify it under some conditions; see the "copying.grx" file
 ** for details.
 **
 ** This library is distributed in the hope that it will be useful,
 ** but WITHOUT ANY WARRANTY; without even the implied warranty of
 ** MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 **
 **/

#include <stdio.h>
#include <string.h>
#include "grx20.h"
#include "grxkeys.h"

#define FONT "../fonts/tms38b.fnt"

int main(void)
{
  char bits[] = {0, 76, 50, 0, 0, 76, 60, 0};
  GrPattern *p1, *p2;
  GrFont *font;
  GrTextOption opt;

  GrSetMode(GR_width_height_color_graphics, 320, 200, (GrColor)256);
  p1 = GrBuildPixmapFromBits(bits, 8, 8, 11,  3);
  p2 = GrBuildPixmapFromBits(bits, 8, 8,  3, 11);
  font = GrLoadFont(FONT);
  if (font && p1 && p2) {
    memset(&opt, 0, sizeof(GrTextOption));
    opt.txo_font   = font;
    opt.txo_xalign = 0;
    opt.txo_yalign = 0;
    opt.txo_direct = GR_TEXT_RIGHT;
    opt.txo_fgcolor.v = GrNOCOLOR;
    opt.txo_bgcolor.v = GrNOCOLOR;
    GrPatternFilledBox(0, 0, GrMaxX(), GrMaxY(), p1);
    GrKeyRead();
    GrPatternDrawString(" Hello world !", 15, 40, 10, &opt, p1);
    GrPatternDrawString(" Hello world !", 15, 44, 50, &opt, p2);
    GrPatternDrawStringExt(" Hello world !!", 16, 40, 100, &opt, p1);
    GrPatternDrawStringExt(" Hello world !!", 16, 44, 140, &opt, p2);
    GrKeyRead();
    opt.txo_bgcolor.v = GrBlack();
    GrPatternDrawString(" Hello world !", 15, 40, 10, &opt, p1);
    GrPatternDrawString(" Hello world !", 15, 44, 50, &opt, p2);
    GrPatternDrawStringExt(" Hello world !!", 16, 40, 100, &opt, p1);
    GrPatternDrawStringExt(" Hello world !!", 16, 44, 140, &opt, p2);
    GrKeyRead();
  }
  if (p1)   GrDestroyPattern(p1);
  if (p2)   GrDestroyPattern(p2);
  if (font) GrUnloadFont(font);
  GrSetMode(GR_default_text);
  if (!p1) fprintf(stderr, "Couldn't create first pattern\n");
  if (!p2) fprintf(stderr, "Couldn't create second pattern\n");
  if (!font) fprintf(stderr, "Couldn't load font %s\n", FONT);

  return 0;
}
```

winclip.c

```
/**
 ** winclip.c ---- clip a drawing to various windows (contexts)
 **
 ** Copyright (c) 1995 Csaba Biegl, 820 Stirrup Dr, Nashville, TN 37221
 ** [e-mail: csaba@vuse.vanderbilt.edu]
 **
 ** This is a test/demo file of the GRX graphics library.
 ** You can use GRX test/demo files as you want.
 **
 ** The GRX graphics library is free software; you can redistribute it
 ** and/or modify it under some conditions; see the "copying.grx" file
 ** for details.
 **
 ** This library is distributed in the hope that it will be useful,
 ** but WITHOUT ANY WARRANTY; without even the implied warranty of
 ** MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 **
 **/

#include "test.h"  /* see [test.h], page 150 */

TESTFUNC(winclip)
```

```
{
int  x = GrSizeX();
int  y = GrSizeY();
int  ww = (x / 2) - 10;
int  wh = (y / 2) - 10;
GrColor c;
GrContext *w1 = GrCreateSubContext(5,5,ww+4,wh+4,NULL,NULL);
GrContext *w2 = GrCreateSubContext(15+ww,5,ww+ww+14,wh+4,NULL,NULL);
GrContext *w3 = GrCreateSubContext(5,15+wh,ww+4,wh+wh+14,NULL,NULL);
GrContext *w4 = GrCreateSubContext(15+ww,15+wh,ww+ww+14,wh+wh+14,NULL,NULL);

GrSetContext(w1);
c = GrAllocColor(200,100,100);
drawing(0,0,ww,wh,c,GrBlack());
GrBox(0,0,ww-1,wh-1,c);

GrSetContext(w2);
c = GrAllocColor(100,200,200);
drawing(-ww/4,ww/3,ww,wh,c,GrBlack());
GrBox(0,0,ww-1,wh-1,c);

GrSetContext(w3);
c = GrAllocColor(200,200,0);
drawing(ww/2,-wh/2,ww,wh,c,GrBlack());
GrBox(0,0,ww-1,wh-1,c);

GrSetContext(w4);
c = GrAllocColor(0,100,200);
drawing(-ww/2,-wh/2,ww*2,wh*2,c,GrBlack());
GrBox(0,0,ww-1,wh-1,c);

GrKeyRead();
}
```

wintest.c

```
/**
 ** wintest.c ---- test window (context) mapping
 **
 ** Copyright (c) 1995 Csaba Biegl, 820 Stirrup Dr, Nashville, TN 37221
 ** [e-mail: csaba@vuse.vanderbilt.edu]
 **
 ** This is a test/demo file of the GRX graphics library.
 ** You can use GRX test/demo files as you want.
 **
 ** The GRX graphics library is free software; you can redistribute it
 ** and/or modify it under some conditions; see the "copying.grx" file
 ** for details.
 **
 ** This library is distributed in the hope that it will be useful,
 ** but WITHOUT ANY WARRANTY; without even the implied warranty of
 ** MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 **
 **/

#include "test.h"  /* see [test.h], page 150 */

TESTFUNC(wintest)
{
int  x = GrSizeX();
int  y = GrSizeY();
int  ww = (x / 2) - 10;
int  wh = (y / 2) - 10;
GrColor c;
GrContext *w1 = GrCreateSubContext(5,5,ww+4,wh+4,NULL,NULL);
GrContext *w2 = GrCreateSubContext(15+ww,5,ww+ww+14,wh+4,NULL,NULL);
GrContext *w3 = GrCreateSubContext(5,15+wh,ww+4,wh+wh+14,NULL,NULL);
GrContext *w4 = GrCreateSubContext(15+ww,15+wh,ww+ww+14,wh+wh+14,NULL,NULL);

GrSetContext(w1);
c = GrAllocColor(200,100,100);
drawing(0,0,ww,wh,c,GrBlack());
GrBox(0,0,ww-1,wh-1,c);

GrSetContext(w2);
c = GrAllocColor(100,200,200);
drawing(0,0,ww,wh,c,GrBlack());
GrBox(0,0,ww-1,wh-1,c);

GrSetContext(w3);
c = GrAllocColor(200,200,0);
drawing(0,0,ww,wh,c,GrBlack());
GrBox(0,0,ww-1,wh-1,c);

GrSetContext(w4);
c = GrAllocColor(0,100,200);
drawing(0,0,ww,wh,c,GrBlack());
GrBox(0,0,ww-1,wh-1,c);

GrKeyRead();
}
```

drawing.h

```
/**
 ** DRAWING.H ---- a stupid little drawing used all over in test programs
 **
 ** Copyright (c) 1995 Csaba Biegl, 820 Stirrup Dr, Nashville, TN 37221
 ** [e-mail: csaba@vuse.vanderbilt.edu] See "doc/copying.cb" for details.
 **/

#include "rand.h"  /* see [rand.h], page 149 */

void drawing(int xpos,int ypos,int xsize,int ysize,long fg,long bg)
{
#   define XP(x)   (int)((((long)(x) * (long)xsize) / 100L) + xpos)
#   define YP(y)   (int)((((long)(y) * (long)ysize) / 100L) + ypos)
int ii;
if(bg != GrNOCOLOR) {
GrFilledBox(xpos,ypos,xpos+xsize-1,ypos+ysize-1,bg);
}
GrLine(XP(10),YP(10),XP(40),YP(40),fg);
GrLine(XP(40),YP(10),XP(10),YP(40),fg);
GrLine(XP(35),YP(10),XP(65),YP(40),fg);
GrLine(XP(35),YP(40),XP(65),YP(10),fg);
GrLine(XP(70),YP(10),XP(90),YP(40),fg);
GrLine(XP(70),YP(40),XP(90),YP(10),fg);
for(ii = 0; ii < 5; ii++) {
GrBox(XP(70+2*ii),YP(10+3*ii),XP(90-2*ii),YP(40-3*ii),fg);
}
GrFilledBox(XP(10),YP(50),XP(60),YP(90),fg);
GrBox(XP(70),YP(50),XP(90),YP(90),fg);
for(ii = 0; ii < 100; ii++) {
GrPlot(XP((RND() % 20U) + 70),YP((RND() % 40U) + 50),fg);
}
}

#undef XP
#undef YP
```

gfaz.h

```
/**
 ** gfaz.h ---- gfaz headers
 **
 ** Copyright (C) 2000,2001 Mariano Alvarez Fernandez
 ** [e-mail: malfer@teleline.es]
 **
 ** This is a test/demo file of the GRX graphics library.
 ** You can use GRX test/demo files as you want.
 **
 ** The GRX graphics library is free software; you can redistribute it
 ** and/or modify it under some conditions; see the "copying.grx" file
 ** for details.
 **
 ** This library is distributed in the hope that it will be useful,
 ** but WITHOUT ANY WARRANTY; without even the implied warranty of
 ** MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 **
 **/

extern GrColor *egacolors;

#define BLACK        egacolors[0]
#define BLUE         egacolors[1]
#define GREEN        egacolors[2]
#define CYAN         egacolors[3]
#define RED          egacolors[4]
#define MAGENTA      egacolors[5]
#define BROWN        egacolors[6]
#define LIGHTGRAY    egacolors[7]
#define DARKGRAY     egacolors[8]
#define LIGHTBLUE    egacolors[9]
#define LIGHTGREEN   egacolors[10]
#define LIGHTCYAN    egacolors[11]
#define LIGHTRED     egacolors[12]
#define LIGHTMAGENTA egacolors[13]
#define YELLOW       egacolors[14]
#define WHITE        egacolors[15]

#define IND_BLACK        0
#define IND_BLUE         1
#define IND_GREEN        2
#define IND_CYAN         3
#define IND_RED          4
#define IND_MAGENTA      5
#define IND_BROWN        6
#define IND_LIGHTGRAY    7
#define IND_DARKGRAY     8
#define IND_LIGHTBLUE    9
#define IND_LIGHTGREEN   10
#define IND_LIGHTCYAN    11
#define IND_LIGHTRED     12
#define IND_LIGHTMAGENTA 13
#define IND_YELLOW       14
#define IND_WHITE        15
```

```
#define EV_NULL    0
#define EV_KEY     1
#define EV_MOUSE   2
#define EV_COMMAND 3
#define EV_SELECT  4
#define EV_END     5

#define MOUSE_LB_PRESSED 1
#define MOUSE_RB_PRESSED 2
#define MOUSE_LB_RELEASED 3
#define MOUSE_RB_RELEASED 4

typedef struct{
  int type;
  long p1;
  long p2;
  long p3;
  } Event;

int gfaz_ini( int width, int height, int bpp );
int gfaz_fin( void );

void event_read( Event *ev );
void event_wait( Event *ev );
void event_queue( Event *ev );
void par_event_queue( int type, long p1, long p2, long p3 );
void set_hook_input_event( void (*fn)( Event * ) );

void show_mouse( void );
void hide_mouse( void );

#define BSTATUS_PRESSED  1
#define BSTATUS_SELECTED 2

typedef struct{
  int x, y;                   // left upper coordinates
  int wide, high;             // what do you mean
  int tbcolor, tfcolor;       // text background, foreground, ind colors
  char *text;                 // the text
  int status;                 // see BSTATUS defs
  int bid;                    // button id
  } Button;

void paint_button( int x, int y, Button *b );

typedef struct{
  int x, y;                   // left upper coordinates
  Button *b;                  // button array
  int nb;                     // button number
  int pb;                     // point actual button
  int abp;                    // actual button pressed
  } Button_Group;

void paint_button_group( Button_Group *bg );
int  pev_button_group( Event *ev, Button_Group *bg );

typedef struct{
  int x, y;                   // left upper coordinates
  int wide, high;             // what do you mean
  int aid;                    // area id
  int divx, divy;             // x, y divisors
  int inip1, inip2;           // par1, par2 initial values
  int incrp1, incrp2;         // par1, par2 increments
  int invert;                 // x,y -> par1,par2 or inverted if 1
  } Area;

typedef struct{
  int x, y;                   // left upper coordinates
  Area *a;                    // area array
  } Area_Group;

int  pev_area_group( Event *ev, Area_Group *ag );

typedef struct{
  int x, y;                   // left upper coordinates
  int wide, high;             // what do you mean
  int lcolor, bcolor, color;  // line, border, normal ind color
  int border;                 //
  } Board;

void paint_board( Board *b );
```

rand.h

```
/**
 ** rand.h ---- a very simple random number generator
 **             (from "Numerical recipies")
 **
 ** This is a test/demo file of the GRX graphics library.
 ** You can use GRX test/demo files as you want.
 **
 ** The GRX graphics library is free software; you can redistribute it
 ** and/or modify it under some conditions; see the "copying.grx" file
 ** for details.
 **
 ** This library is distributed in the hope that it will be useful,
 ** but WITHOUT ANY WARRANTY; without even the implied warranty of
 ** MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 **
 **/

#ifndef __RAND_H_INCLUDED
#define __RAND_H_INCLUDED

#define _IA   16807
#define _IM   2147483647L
#define _IQ   127773L
#define _IR   2836
#define _MASK 123459876UL

static long _idum = 0;

unsigned long ran0(void) {
  long k;
  _idum ^= _MASK;
  k = _idum / _IQ;
  _idum = _IA * (_idum - k * _IQ) - _IR * k;
  if (_idum < 0) _idum += _IM;
  return (unsigned long) _idum;
}

#define sran0(x) do _idum = (x); while(0)

#define RND()    ran0()
#define SRND(x)  sran0(x)
#define RND_MAX  (_MASK)

#endif
```

test.h

```
/**
 ** test.h ---- common declarations for test programs
 **
 ** Copyright (c) 1995 Csaba Biegl, 820 Stirrup Dr, Nashville, TN 37221
 ** [e-mail: csaba@vuse.vanderbilt.edu]
 **
 ** This is a test/demo file of the GRX graphics library.
 ** You can use GRX test/demo files as you want.
 **
 ** The GRX graphics library is free software; you can redistribute it
 ** and/or modify it under some conditions; see the "copying.grx" file
 ** for details.
 **
 ** This library is distributed in the hope that it will be useful,
 ** but WITHOUT ANY WARRANTY; without even the implied warranty of
 ** MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 **
 **/

#ifndef __TEST_H_INCLUDED__
#define __TEST_H_INCLUDED__

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "grx20.h"
#include "grxkeys.h"
#include "drawing.h"  /* see [drawing.h], page 146 */

extern void (*testfunc)(void);
char   exit_message[2000] = { "" };
int    Argc;
char **Argv;

#define TESTFUNC(name)      \
void name(void);        \
void (*testfunc)(void) = name;  \
void name(void)

int main(int argc,char **argv)
{
int  x = 0;
int  y = 0;
long c = 0;
char m[41];

Argc = argc - 1;
Argv = argv + 1;
if((Argc >= 2) &&
   (sscanf(Argv[0],"%d",&x) == 1) && (x >= 320) &&
   (sscanf(Argv[1],"%d",&y) == 1) && (y >= 200)) {
Argc -= 2;
Argv += 2;
if (Argc > 0) {
   char *endp;
   c = strtol(Argv[0], &endp, 0);
   switch (*endp) {
     case 'k':
     case 'K': c <<= 10; break;
     case 'm':
     case 'M': c <<= 20; break;
   }
   Argc--;
   Argv++;
}
}
if(c >= 2)
GrSetMode(GR_width_height_color_graphics,x,y,c);
else if((x >= 320) && (y >= 200))
GrSetMode(GR_width_height_graphics,x,y);
else GrSetMode(GR_default_graphics);
(*testfunc)();
GrSetMode(GR_default_text);
if(strlen(exit_message) > 0) {
puts(exit_message);
fflush(stdout);
fgets(m,40,stdin);
}
return(0);
}

#endif /* _TEST_H_ */
```

arctest.dat

```
arc xc=300 yc=200 xa=50  ya=50  start=10   end=40
arc xc=300 yc=200 xa=250 ya=150 start=10   end=200
arc xc=300 yc=200 xa=250 ya=150 start=10   end=2000
arc xc=300 yc=200 xa=250 ya=150 start=1000 end=200
arc xc=300 yc=200 xa=25  ya=15  start=3500 end=800
arc xc=300 yc=200 xa=25  ya=15  start=10   end=100
arc xc=300 yc=200 xa=25  ya=15  start=3500 end=10
arc xc=300 yc=200 xa=25  ya=15  start=0    end=900
```

polytest.dat

```
beginc
300 200
400 400
200 400
```

```
end

beginc
300 200
400 400
150 470
200 400
end

beginc
300 200
400 400
150 470
120 330
end

beginc
300 200
400 400
050 470
020 330
end

beginc
300 -100
400 400
050 870
020 330
end

beginc
300 20
400 100
050 100
end

beginc
400 500
050 500
200 560
end

beginc
400 500
250 495
050 500
200 560
```

```
end

beginc
100 500
400 500
300 550
end

beginc
150 150
300 150
250 250
300 400
120 444
end

begin
250 150
200 450
350 250
150 250
400 450
end

beginc
150 150
400 200
400 202
end

begin
-10 0
100 -10
200 200
-10 200
end
```

Includes

grx20.h

```
/**
 ** grx20.h ---- GRX 2.x API functions and data structure declarations
 **
 ** Copyright (c) 1995 Csaba Biegl, 820 Stirrup Dr, Nashville, TN 37221
 ** [e-mail: csaba@vuse.vanderbilt.edu]
 **
 ** This file is part of the GRX graphics library.
 **
 ** The GRX graphics library is free software; you can redistribute it
 ** and/or modify it under some conditions; see the "copying.grx" file
 ** for details.
 **
 ** This library is distributed in the hope that it will be useful,
 ** but WITHOUT ANY WARRANTY; without even the implied warranty of
 ** MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 **
 **/

#ifndef __GRX20_H_INCLUDED__
#define __GRX20_H_INCLUDED__

/* ================================================================== */
/*       COMPILER -- CPU -- SYSTEM SPECIFIC VERSION STUFF             */
/* ================================================================== */

/* Version of GRX API
**
** usage:
**    #include <grx20.h>
**    #ifndef GRX_VERSION_API
**    #ifdef  GRX_VERSION
**    #define GRX_VERSION_API 0x0200
**    #else
**    #define GRX_VERSION_API 0x0103
**    #endif
**    #endif
*/
#define GRX_VERSION_API 0x0247

/* these are the supported configurations: */
#define GRX_VERSION_TCC_8086_DOS        1       /* also works with BCC */
#define GRX_VERSION_GCC_386_GO32        2       /* deprecated, don't use it */
#define GRX_VERSION_GCC_386_DJGPP       2       /* DJGPP v2 */
#define GRX_VERSION_GCC_386_LINUX       3       /* the real stuff */
#define GRX_VERSION_GENERIC_X11         4       /* generic X11 version */
#define GRX_VERSION_WATCOM_DOS4GW       5       /* GS - Watcom C++ 11.0 32 Bit */
/*#define GRX_VERSION_WATCOM_REAL_MODE  6*/     /* GS - Watcom C++ 11.0 16 Bit -
TODO! */
#define GRX_VERSION_GCC_386_WIN32       7       /* WIN32 using Mingw32 */
#define GRX_VERSION_MSC_386_WIN32       8       /* WIN32 using MS-VC */
#define GRX_VERSION_GCC_386_CYG32       9       /* WIN32 using CYGWIN */
#define GRX_VERSION_GCC_386_X11        10       /* X11 version */
#define GRX_VERSION_GCC_X86_64_LINUX   11       /* console framebuffer 64 */
```

```
#define GRX_VERSION_GCC_X86_64_X11     12       /* X11 version 64 */

#define GRXMain main  /* From the 2.4.6 version We don't need this */
                      /* anymore, but it is here for previous apps */

#ifdef  __TURBOC__
#define GRX_VERSION     GRX_VERSION_TCC_8086_DOS
#endif

#ifdef  __GNUC__
#ifdef  __DJGPP__
#define GRX_VERSION     GRX_VERSION_GCC_386_DJGPP
#endif
#if defined(__XWIN__)
#if defined(__linux__) && defined(__i386__)
#define GRX_VERSION     GRX_VERSION_GCC_386_X11
#endif
#if defined(__linux__) && defined(__x86_64__)
#define GRX_VERSION     GRX_VERSION_GCC_X86_64_X11
#endif
#else
#if defined(__linux__) && defined(__i386__)
#define GRX_VERSION     GRX_VERSION_GCC_386_LINUX
#endif
#if defined(__linux__) && defined(__x86_64__)
#define GRX_VERSION     GRX_VERSION_GCC_X86_64_LINUX
#endif
#endif
#ifdef  __WIN32__
#define GRX_VERSION     GRX_VERSION_GCC_386_WIN32
#endif
#ifdef __CYGWIN32__
#define GRX_VERSION     GRX_VERSION_GCC_386_CYG32
#define __WIN32__
#endif
#endif /* __GNUC__ */

#ifdef  __WATCOMC__     /* GS - Watcom C++ 11.0 */
#ifdef  __DOS__
#ifdef  __386__
#define GRX_VERSION     GRX_VERSION_WATCOM_DOS4GW
#else
/* #define GRX_VERSION GRX_VERSION_WATCOM_REAL_MODE  - I haven't tested GRX in 16 bit*
#endif /* __386__ */
#endif /* __DOS__ */
#endif /* __WATCOMC__ */
```

```
#ifdef _MSC_VER
#ifdef _WIN32
#ifdef _M_IX86
#define GRX_VERSION     GRX_VERSION_MSC_386_WIN32
#if !defined(__WIN32__)
#define __WIN32__ _WIN32
#endif
#endif /* _M_IX86  */
#endif /* _WIN32   */
#endif /* _MSC_VER */

#ifndef GRX_VERSION
#if defined(unix) || defined(__unix) || defined(__unix__) || defined(_AIX)
#define GRX_VERSION     GRX_VERSION_GENERIC_X11
#endif
#endif

#ifndef GRX_VERSION
#error  GRX is not supported on your COMPILER/CPU/OPERATING SYSTEM!
#endif

#if (GRX_VERSION==GRX_VERSION_WATCOM_DOS4GW)
#define near
#define far
#define huge
#endif

#if !defined(__TURBOC__) && (GRX_VERSION!=GRX_VERSION_WATCOM_REAL_MODE)
#ifndef near            /* get rid of these stupid keywords */
#define near
#endif
#ifndef far
#define far
#endif
#ifndef huge
#define huge
#endif
#endif

#ifdef __cplusplus
extern "C" {
#endif

/* a couple of forward declarations ... */
typedef struct _GR_frameDriver  GrFrameDriver;
typedef struct _GR_videoDriver  GrVideoDriver;
typedef struct _GR_videoMode    GrVideoMode;
typedef struct _GR_videoModeExt GrVideoModeExt;
typedef struct _GR_frame        GrFrame;
typedef struct _GR_context      GrContext;

/* ================================================================== */
/*                        SYSTEM TYPE DEF's                           */
/* ================================================================== */

/* need unsigned 32 bit integer for color stuff */
#if defined(__TURBOC__) && defined(__MSDOS__)
/* TCC && BCC are 16 bit compilers */
typedef unsigned long int GrColor;
#else
/* all other platforms (GCC on i386 or x86_64 and ALPHA) have 32 bit ints */
typedef unsigned int GrColor;
#endif

/* ================================================================== */
/*                           MODE SETTING                             */
/* ================================================================== */

/*
 * available video modes (for 'GrSetMode')
 */
typedef enum _GR_graphicsModes {
GR_unknown_mode = (-1),             /* initial state */
/* ============= modes which clear the video memory ============= */
GR_80_25_text = 0,                  /* Extra parameters for GrSetMode: */
GR_default_text,
GR_width_height_text,               /* int w,int h */
GR_biggest_text,
GR_320_200_graphics,
GR_default_graphics,
GR_width_height_graphics,           /* int w,int h */
GR_biggest_noninterlaced_graphics,
GR_biggest_graphics,
GR_width_height_color_graphics,     /* int w,int h,GrColor nc */
GR_width_height_color_text,         /* int w,int h,GrColor nc */
GR_custom_graphics,                 /* int w,int h,GrColor nc,int vx,int vy */
/* ==== equivalent modes which do not clear the video memory ==== */
GR_NC_80_25_text,
GR_NC_default_text,
GR_NC_width_height_text,            /* int w,int h */
GR_NC_biggest_text,
GR_NC_320_200_graphics,
GR_NC_default_graphics,
GR_NC_width_height_graphics,        /* int w,int h */
```

```
GR_NC_biggest_noninterlaced_graphics,
GR_NC_biggest_graphics,
GR_NC_width_height_color_graphics,  /* int w,int h,GrColor nc */
GR_NC_width_height_color_text,      /* int w,int h,GrColor nc */
GR_NC_custom_graphics,              /* int w,int h,GrColor nc,int vx,int vy */
/* ==== plane instead of color based modes ==== */
/* colors = 1 << bpp  >>> resort enum for GRX3 <<< */
GR_width_height_bpp_graphics,       /* int w,int h,int bpp */
GR_width_height_bpp_text,           /* int w,int h,int bpp */
GR_custom_bpp_graphics,             /* int w,int h,int bpp,int vx,int vy */
GR_NC_width_height_bpp_graphics,    /* int w,int h,int bpp */
GR_NC_width_height_bpp_text,        /* int w,int h,int bpp */
GR_NC_custom_bpp_graphics           /* int w,int h,int bpp,int vx,int vy */
} GrGraphicsMode;

/*
 * Available frame modes (video memory layouts)
 */
typedef enum _GR_frameModes {
/* ====== video frame buffer modes ====== */
GR_frameUndef,                      /* undefined */
GR_frameText,                       /* text modes */
GR_frameHERC1,                      /* Hercules mono */
GR_frameEGAVGA1,                    /* EGA VGA mono */
GR_frameEGA4,                       /* EGA 16 color */
GR_frameSVGA4,                      /* (Super) VGA 16 color */
GR_frameSVGA8,                      /* (Super) VGA 256 color */
GR_frameVGA8X,                      /* VGA 256 color mode X */
GR_frameSVGA16,                     /* Super VGA 32768/65536 color */
GR_frameSVGA24,                     /* Super VGA 16M color */
GR_frameSVGA32L,                    /* Super VGA 16M color padded #1 */
GR_frameSVGA32H,                    /* Super VGA 16M color padded #2 */
/* ==== modes provided by the X11 driver ===== */
GR_frameXWIN1   = GR_frameEGAVGA1,
GR_frameXWIN4   = GR_frameSVGA4,
GR_frameXWIN8   = GR_frameSVGA8,
GR_frameXWIN16  = GR_frameSVGA16,
GR_frameXWIN24  = GR_frameSVGA24,
GR_frameXWIN32L = GR_frameSVGA32L,
GR_frameXWIN32H = GR_frameSVGA32H,
/* ==== modes provided by the WIN32 driver ===== */
GR_frameWIN32_1   = GR_frameEGAVGA1,
GR_frameWIN32_4   = GR_frameSVGA4,
GR_frameWIN32_8   = GR_frameSVGA8,
GR_frameWIN32_16  = GR_frameSVGA16,
GR_frameWIN32_24  = GR_frameSVGA24,
GR_frameWIN32_32L = GR_frameSVGA32L,
GR_frameWIN32_32H = GR_frameSVGA32H,
/* ==== modes provided by the SDL driver ===== */
GR_frameSDL8   = GR_frameSVGA8,
GR_frameSDL16  = GR_frameSVGA16,
GR_frameSDL24  = GR_frameSVGA24,
GR_frameSDL32L = GR_frameSVGA32L,
GR_frameSDL32H = GR_frameSVGA32H,
/* ==== linear frame buffer modes  ====== */
GR_frameSVGA8_LFB,                   /* (Super) VGA 256 color */
GR_frameSVGA16_LFB,                  /* Super VGA 32768/65536 color */
GR_frameSVGA24_LFB,                  /* Super VGA 16M color */
GR_frameSVGA32L_LFB,                 /* Super VGA 16M color padded #1 */
GR_frameSVGA32H_LFB,                 /* Super VGA 16M color padded #2 */
/* ====== system RAM frame buffer modes ====== */
GR_frameRAM1,                        /* mono */
GR_frameRAM4,                        /* 16 color planar */
GR_frameRAM8,                        /* 256 color */
GR_frameRAM16,                       /* 32768/65536 color */
GR_frameRAM24,                       /* 16M color */
GR_frameRAM32L,                      /* 16M color padded #1 */
GR_frameRAM32H,                      /* 16M color padded #2 */
GR_frameRAM3x8,                      /* 16M color planar (image mode) */
/* ====== markers for scanning modes ====== */
GR_firstTextFrameMode     = GR_frameText,
GR_lastTextFrameMode      = GR_frameText,
GR_firstGraphicsFrameMode = GR_frameHERC1,
GR_lastGraphicsFrameMode  = GR_frameSVGA32H_LFB,
GR_firstRAMframeMode      = GR_frameRAM1,
GR_lastRAMframeMode       = GR_frameRAM3x8
} GrFrameMode;

/*
 * supported video adapter types
 */
typedef enum _GR_videoAdapters {
GR_UNKNOWN = (-1),                   /* not known (before driver set) */
GR_VGA,                              /* VGA adapter */
GR_EGA,                              /* EGA adapter */
GR_HERC,                             /* Hercules mono adapter */
GR_8514A,                            /* 8514A or compatible */
GR_S3,                               /* S3 graphics accelerator */
GR_XWIN,                             /* X11 driver */
GR_WIN32,                            /* WIN32 driver */
GR_LNXFB,                            /* Linux framebuffer */
GR_SDL,                              /* SDL driver */
GR_MEM                               /* memory only driver */
} GrVideoAdapter;
```

```
/*
 * The video driver descriptor structure
 */
struct _GR_videoDriver {
char   *name;                       /* driver name */
enum   _GR_videoAdapters adapter;   /* adapter type */
struct _GR_videoDriver  *inherit;   /* inherit video modes from this */
struct _GR_videoMode    *modes;     /* table of supported modes */
int    nmodes;                      /* number of modes */
int   (*detect)(void);
int   (*init)(char *options);
void  (*reset)(void);
GrVideoMode * (*selectmode)(GrVideoDriver *drv,int w,int h,int bpp,
int txt,unsigned int *ep);
unsigned  drvflags;
};
/* bits in the drvflags field: */
#define GR_DRIVERF_USER_RESOLUTION 1
  /* set if driver supports user setable arbitrary resolution */


/*
 * Video driver mode descriptor structure
 */
struct _GR_videoMode {
char    present;                    /* is it really available? */
char    bpp;                        /* log2 of # of colors */
short   width,height;               /* video mode geometry */
short   mode;                       /* BIOS mode number (if any) */
int     lineoffset;                 /* scan line length */
int     privdata;                   /* driver can use it for anything */
struct _GR_videoModeExt *extinfo;   /* extra info (maybe shared) */
};

/*
 * Video driver mode descriptor extension structure. This is a separate
 * structure accessed via a pointer from the main mode descriptor. The
 * reason for this is that frequently several modes can share the same
 * extended info.
 */
struct _GR_videoModeExt {
enum   _GR_frameModes   mode;       /* frame driver for this video mode */█
struct _GR_frameDriver *drv;        /* optional frame driver override */
char    far *frame;                 /* frame buffer address */
char    cprec[3];                   /* color component precisions */
char    cpos[3];                    /* color component bit positions */
```

```
int     flags;                      /* mode flag bits; see "grdriver.h" */
int   (*setup)(GrVideoMode *md,int noclear);
int   (*setvsize)(GrVideoMode *md,int w,int h,GrVideoMode *result);
int   (*scroll)(GrVideoMode *md,int x,int y,int result[2]);
void  (*setbank)(int bk);
void  (*setrwbanks)(int rb,int wb);
void  (*loadcolor)(int c,int r,int g,int b);
int     LFB_Selector;
};

/*
 * The frame driver descriptor structure.
 */
struct _GR_frameDriver {
    enum    _GR_frameModes mode;         /* supported frame access mode */
    enum    _GR_frameModes rmode;        /* matching RAM frame (if video) */
    int      is_video;                   /* video RAM frame driver ? */
    int      row_align;                  /* scan line size alignment */
    int      num_planes;                 /* number of planes */
    int      bits_per_pixel;             /* bits per pixel */
    long     max_plane_size;             /* maximum plane size in bytes */
    int      (*init)(GrVideoMode *md);
    GrColor  (*readpixel)(GrFrame *c,int x,int y);
    void     (*drawpixel)(int x,int y,GrColor c);
    void     (*drawline)(int x,int y,int dx,int dy,GrColor c);
    void     (*drawhline)(int x,int y,int w,GrColor c);
    void     (*drawvline)(int x,int y,int h,GrColor c);
    void     (*drawblock)(int x,int y,int w,int h,GrColor c);
    void     (*drawbitmap)(int x,int y,int w,int h,char far *bmp,int pitch,int start,G
    void     (*drawpattern)(int x,int y,int w,char patt,GrColor fg,GrColor bg);
    void     (*bitblt)(GrFrame *dst,int dx,int dy,GrFrame *src,int x,int y,int w,int h
    void     (*bltv2r)(GrFrame *dst,int dx,int dy,GrFrame *src,int x,int y,int w,int h
    void     (*bltr2v)(GrFrame *dst,int dx,int dy,GrFrame *src,int x,int y,int w,int h
    /* new functions in v2.3 */
    GrColor far *(*getindexedscanline)(GrFrame *c,int x, int y, int w, int *indx);
      /* will return an array of pixel values pv[] read from frame   */
      /*    if indx == NULL: pv[i=0..w-1] = readpixel(x+i,y)         */
      /*    else             pv[i=0..w-1] = readpixel(x+indx[i],y)   */
    void     (*putscanline)(int x, int y, int w,const GrColor far *scl, Gr-
Color op);
      /** will draw scl[i=0..w-1] to frame:                          */
      /*    if (scl[i] != skipcolor) drawpixel(x+i,y,(scl[i] | op))  */
};

/*
 * driver and mode info structure
 */
```

```
extern const struct _GR_driverInfo {
struct _GR_videoDriver  *vdriver;   /* the current video driver */
struct _GR_videoMode    *curmode;   /* current video mode pointer */
struct _GR_videoMode     actmode;   /* copy of above, resized if virtual */
struct _GR_frameDriver   fdriver;   /* frame driver for the current con-
text */
struct _GR_frameDriver   sdriver;   /* frame driver for the screen */
struct _GR_frameDriver   tdriver;   /* a dummy driver for text modes */
enum   _GR_graphicsModes mcode;     /* code for the current mode */
int     deftw,defth;                /* default text mode size */
int     defgw,defgh;                /* default graphics mode size */
GrColor deftc,defgc;                /* default text and graphics colors */
int     vposx,vposy;                /* current virtual viewport position */
int     errsfatal;                  /* if set, exit upon errors */
int     moderestore;                /* restore startup video mode if set */
int     splitbanks;                 /* indicates separate R/W banks */
int     curbank;                    /* currently mapped bank */
void  (*mdsethook)(void);           /* callback for mode set */
void  (*setbank)(int bk);           /* banking routine */
void  (*setrwbanks)(int rb,int wb); /* split banking routine */
} * const GrDriverInfo;

/*
 * setup stuff
 */
int  GrSetDriver(char *drvspec);
int  GrSetMode(GrGraphicsMode which,...);
int  GrSetViewport(int xpos,int ypos);
void GrSetModeHook(void (*hookfunc)(void));
void GrSetModeRestore(int restoreFlag);
void GrSetErrorHandling(int exitIfError);
void GrSetEGAVGAmonoDrawnPlane(int plane);
void GrSetEGAVGAmonoShownPlane(int plane);

unsigned GrGetLibraryVersion(void);
unsigned GrGetLibrarySystem(void);

/*
 * inquiry stuff ---- many of these are actually macros (see below)
 */
GrGraphicsMode GrCurrentMode(void);
GrVideoAdapter GrAdapterType(void);
GrFrameMode    GrCurrentFrameMode(void);
GrFrameMode    GrScreenFrameMode(void);
GrFrameMode    GrCoreFrameMode(void);

const GrVideoDriver *GrCurrentVideoDriver(void);
const GrVideoMode   *GrCurrentVideoMode(void);
const GrVideoMode   *GrVirtualVideoMode(void);
const GrFrameDriver *GrCurrentFrameDriver(void);
const GrFrameDriver *GrScreenFrameDriver(void);
const GrVideoMode   *GrFirstVideoMode(GrFrameMode fmode);
const GrVideoMode   *GrNextVideoMode(const GrVideoMode *prev);

int  GrScreenX(void);
int  GrScreenY(void);
int  GrVirtualX(void);
int  GrVirtualY(void);
int  GrViewportX(void);
int  GrViewportY(void);

int  GrScreenIsVirtual(void);

/*
 * RAM context geometry and memory allocation inquiry stuff
 */
int  GrFrameNumPlanes(GrFrameMode md);
int  GrFrameLineOffset(GrFrameMode md,int width);
long GrFramePlaneSize(GrFrameMode md,int w,int h);
long GrFrameContextSize(GrFrameMode md,int w,int h);

int  GrNumPlanes(void);
int  GrLineOffset(int width);
long GrPlaneSize(int w,int h);
long GrContextSize(int w,int h);

/*
 * inline implementation for some of the above
 */
#ifndef GRX_SKIP_INLINES
#define GrAdapterType()         (GrDriverInfo->vdriver ? GrDriverInfo->vdriver-
>adapter : GR_UNKNOWN)
#define GrCurrentMode()         (GrDriverInfo->mcode)
#define GrCurrentFrameMode()    (GrDriverInfo->fdriver.mode)
#define GrScreenFrameMode()     (GrDriverInfo->sdriver.mode)
#define GrCoreFrameMode()       (GrDriverInfo->sdriver.rmode)

#define GrCurrentVideoDriver()  ((const GrVideoDriver *)( GrDriverInfo-
>vdriver))
#define GrCurrentVideoMode()    ((const GrVideoMode   *)( GrDriverInfo-
>curmode))
#define GrVirtualVideoMode()    ((const GrVideoMode   *)(&GrDriverInfo-
>actmode))
```

```
#define GrCurrentFrameDriver()  ((const GrFrameDriver *)(&GrDriverInfo-
>fdriver))
#define GrScreenFrameDriver()   ((const GrFrameDriver *)(&GrDriverInfo-
>sdriver))

#define GrIsFixedMode()      (!(  GrCurrentVideoDriver()->drvflags \
   & GR_DRIVERF_USER_RESOLUTION))

#define GrScreenX()             (GrCurrentVideoMode()->width)
#define GrScreenY()             (GrCurrentVideoMode()->height)
#define GrVirtualX()            (GrVirtualVideoMode()->width)
#define GrVirtualY()            (GrVirtualVideoMode()->height)
#define GrViewportX()           (GrDriverInfo->vposx)
#define GrViewportY()           (GrDriverInfo->vposy)

#define GrScreenIsVirtual()     ((GrScreenX() + GrScreenY()) < (GrVirtu-
alX() + GrVirtualY()))

#define GrNumPlanes()           GrFrameNumPlanes(GrCoreFrameMode())
#define GrLineOffset(w)         GrFrameLineOffset(GrCoreFrameMode(),w)
#define GrPlaneSize(w,h)        GrFramePlaneSize(GrCoreFrameMode(),w,h)
#define GrContextSize(w,h)      GrFrameContextSize(GrCoreFrameMode(),w,h)
#endif  /* GRX_SKIP_INLINES */


/* ================================================================== */
/*              FRAME BUFFER, CONTEXT AND CLIPPING STUFF              */
/* ================================================================== */

struct _GR_frame {
char    far *gf_baseaddr[4];        /* base address of frame memory */
short   gf_selector;                /* frame memory segment selector */
char    gf_onscreen;                /* is it in video memory ? */
char    gf_memflags;                /* memory allocation flags */
int     gf_lineoffset;              /* offset to next scan line in bytes */
struct _GR_frameDriver *gf_driver;  /* frame access functions */
};

struct _GR_context {
struct _GR_frame    gc_frame;       /* frame buffer info */
struct _GR_context *gc_root;        /* context which owns frame */
int    gc_xmax;                     /* max X coord (width  - 1) */
int    gc_ymax;                     /* max Y coord (height - 1) */
int    gc_xoffset;                  /* X offset from root's base */
int    gc_yoffset;                  /* Y offset from root's base */
int    gc_xcliplo;                  /* low X clipping limit */
int    gc_ycliplo;                  /* low Y clipping limit */
int    gc_xcliphi;                  /* high X clipping limit */
int    gc_ycliphi;                  /* high Y clipping limit */
int    gc_usrxbase;                 /* user window min X coordinate */
int    gc_usrybase;                 /* user window min Y coordinate */
int    gc_usrwidth;                 /* user window width  */
int    gc_usrheight;                /* user window height */
#   define gc_baseaddr                  gc_frame.gf_baseaddr
#   define gc_selector                  gc_frame.gf_selector
#   define gc_onscreen                  gc_frame.gf_onscreen
#   define gc_memflags                  gc_frame.gf_memflags
#   define gc_lineoffset                gc_frame.gf_lineoffset
#   define gc_driver                    gc_frame.gf_driver
};

extern const struct _GR_contextInfo {
struct _GR_context current;         /* the current context */
struct _GR_context screen;          /* the screen context */
} * const GrContextInfo;

GrContext *GrCreateContext(int w,int h,char far *memory[4],GrContext *where);
GrContext *GrCreateFrameContext(GrFrameMode md,int w,int h,char far *mem-
ory[4],GrContext *where);
GrContext *GrCreateSubContext(int x1,int y1,int x2,int y2,const GrContext *par-
ent,GrContext *where);
GrContext *GrSaveContext(GrContext *where);

GrContext *GrCurrentContext(void);
GrContext *GrScreenContext(void);

void  GrDestroyContext(GrContext *context);
void  GrResizeSubContext(GrContext *context,int x1,int y1,int x2,int y2);
void  GrSetContext(const GrContext *context);

void  GrSetClipBox(int x1,int y1,int x2,int y2);
void  GrSetClipBoxC(GrContext *c,int x1,int y1,int x2,int y2);
void  GrGetClipBox(int *x1p,int *y1p,int *x2p,int *y2p);
void  GrGetClipBoxC(const GrContext *c,int *x1p,int *y1p,int *x2p,int *y2p);
void  GrResetClipBox(void);
void  GrResetClipBoxC(GrContext *c);

int   GrMaxX(void);
int   GrMaxY(void);
int   GrSizeX(void);
int   GrSizeY(void);
int   GrLowX(void);
int   GrLowY(void);
int   GrHighX(void);
int   GrHighY(void);

#ifndef GRX_SKIP_INLINES
#define GrCreateContext(w,h,m,c) (GrCreateFrameContext(GrCoreFrameMode(),w,h,m,c))
#define GrCurrentContext()       ((GrContext *)(&GrContextInfo->current))
#define GrScreenContext()        ((GrContext *)(&GrContextInfo->screen))
#define GrMaxX()                 (GrCurrentContext()->gc_xmax)
#define GrMaxY()                 (GrCurrentContext()->gc_ymax)
#define GrSizeX()                (GrMaxX() + 1)
#define GrSizeY()                (GrMaxY() + 1)
#define GrLowX()                 (GrCurrentContext()->gc_xcliplo)
#define GrLowY()                 (GrCurrentContext()->gc_ycliplo)
#define GrHighX()                (GrCurrentContext()->gc_xcliphi)
#define GrHighY()                (GrCurrentContext()->gc_ycliphi)
#define GrGetClipBoxC(C,x1p,y1p,x2p,y2p) do {           \
*(x1p) = (C)->gc_xcliplo;                           \
*(y1p) = (C)->gc_ycliplo;                           \
*(x2p) = (C)->gc_xcliphi;                           \
*(y2p) = (C)->gc_ycliphi;                           \
} while(0)
#define GrGetClipBox(x1p,y1p,x2p,y2p) do {              \
*(x1p) = GrLowX();                                  \
*(y1p) = GrLowY();                                  \
*(x2p) = GrHighX();                                 \
*(y2p) = GrHighY();                                 \
} while(0)
#endif  /* GRX_SKIP_INLINES */

/* =================================================================== */
/*                            COLOR STUFF                              */
/* =================================================================== */

/*
 * Flags to 'OR' to colors for various operations
 */
#define GrWRITE         0UL             /* write color */
#define GrXOR           0x01000000UL    /* to "XOR" any color to the screen */
#define GrOR            0x02000000UL    /* to "OR" to the screen */
#define GrAND           0x03000000UL    /* to "AND" to the screen */
#define GrIMAGE         0x04000000UL    /* BLIT: write, except given color */
#define GrCVALUEMASK    0x00ffffffUL    /* color value mask */
#define GrCMODEMASK     0xff000000UL    /* color operation mask */
#define GrNOCOLOR       (GrXOR | 0)     /* GrNOCOLOR is used for "no" color */

GrColor GrColorValue(GrColor c);
GrColor GrColorMode(GrColor c);
GrColor GrWriteModeColor(GrColor c);
GrColor GrXorModeColor(GrColor c);
GrColor GrOrModeColor(GrColor c);
GrColor GrAndModeColor(GrColor c);
GrColor GrImageModeColor(GrColor c);

/*
 * color system info structure (all [3] arrays are [r,g,b])
 */
extern const struct _GR_colorInfo {
GrColor       ncolors;              /* number of colors */
GrColor       nfree;                /* number of unallocated colors */
GrColor       black;                /* the black color */
GrColor       white;                /* the white color */
unsigned int  RGBmode;              /* set when RGB mode */
unsigned int  prec[3];              /* color field precisions */
unsigned int  pos[3];               /* color field positions */
unsigned int  mask[3];              /* masks for significant bits */
unsigned int  round[3];             /* add these for rounding */
unsigned int  shift[3];             /* shifts for (un)packing color */
unsigned int  norm;                 /* normalization for (un)packing */
struct {                            /* color table for non-RGB modes */
unsigned char r,g,b;            /* loaded components */
unsigned int  defined:1;        /* r,g,b values are valid if set */
unsigned int  writable:1;       /* can be changed by 'GrSetColor' */
unsigned long int nused;        /* usage count */
} ctable[256];
} * const GrColorInfo;

void    GrResetColors(void);
void    GrSetRGBcolorMode(void);
void    GrRefreshColors(void);

GrColor GrNumColors(void);
GrColor GrNumFreeColors(void);

GrColor GrBlack(void);
GrColor GrWhite(void);

GrColor GrBuildRGBcolorT(int r,int g,int b);
GrColor GrBuildRGBcolorR(int r,int g,int b);
int     GrRGBcolorRed(GrColor c);
int     GrRGBcolorGreen(GrColor c);
int     GrRGBcolorBlue(GrColor c);

GrColor GrAllocColor(int r,int g,int b);   /* shared, read-only */
GrColor GrAllocColorID(int r,int g,int b); /* potentially inlined version */
GrColor GrAllocColor2(long hcolor);        /* shared, read-only, 0xRRGGBB */
GrColor GrAllocColor2ID(long hcolor);      /* potentially inlined version */
GrColor GrAllocCell(void);                 /* unshared, read-write */

GrColor *GrAllocEgaColors(void);           /* shared, read-only standard EGA col-
ors */

void    GrSetColor(GrColor c,int r,int g,int b);
void    GrFreeColor(GrColor c);
void    GrFreeCell(GrColor c);

void    GrQueryColor(GrColor c,int *r,int *g,int *b);
void    GrQueryColorID(GrColor c,int *r,int *g,int *b);
void    GrQueryColor2(GrColor c,long *hcolor);
void    GrQueryColor2ID(GrColor c,long *hcolor);

int     GrColorSaveBufferSize(void);
void    GrSaveColors(void *buffer);
void    GrRestoreColors(void *buffer);

#ifndef GRX_SKIP_INLINES
#define GrColorValue(c)         ((GrColor)(c) & GrCVALUEMASK)
#define GrColorMode(c)          ((GrColor)(c) & GrCMODEMASK)
#define GrWriteModeColor(c)     (GrColorValue(c) | GrWRITE)
#define GrXorModeColor(c)       (GrColorValue(c) | GrXOR)
#define GrOrModeColor(c)        (GrColorValue(c) | GrOR)
#define GrAndModeColor(c)       (GrColorValue(c) | GrAND)
#define GrImageModeColor(c)     (GrColorValue(c) | GrIMAGE)
#define GrNumColors()           (GrColorInfo->ncolors)
#define GrNumFreeColors()       (GrColorInfo->nfree)
#define GrBlack() (                                                            \
(GrColorInfo->black == GrNOCOLOR) ?                                    \
(GrBlack)() :                                                          \
GrColorInfo->black                                                     \
)
#define GrWhite() (                                                            \
(GrColorInfo->white == GrNOCOLOR) ?                                    \
(GrWhite)() :                                                          \
GrColorInfo->white                                                     \
)
#define GrBuildRGBcolorT(r,g,b) ((                                             \
((GrColor)((int)(r) & GrColorInfo->mask[0]) << GrColorInfo->shift[0]) |\
((GrColor)((int)(g) & GrColorInfo->mask[1]) << GrColorInfo->shift[1]) |\
((GrColor)((int)(b) & GrColorInfo->mask[2]) << GrColorInfo->shift[2])  \
) >> GrColorInfo->norm                                                 \
)
#define GrBuildRGBcolorR(r,g,b) GrBuildRGBcolorT(                              \
(((unsigned int)(r)) > GrColorInfo->mask[0]) ? 255 : (unsigned int)(r) + GrColorInfo-
>round[0], \
(((unsigned int)(g)) > GrColorInfo->mask[1]) ? 255 : (unsigned int)(g) + GrColorInfo-
>round[1], \
(((unsigned int)(b)) > GrColorInfo->mask[2]) ? 255 : (unsigned int)(b) + GrColorInfo-
>round[2]  \
)
#define GrRGBcolorRed(c) (                                                    \
(int)(((GrColor)(c) << GrColorInfo->norm) >> GrColorInfo->shift[0]) &  \
(GrColorInfo->mask[0])                                                 \
)
#define GrRGBcolorGreen(c) (                                                  \
(int)(((GrColor)(c) << GrColorInfo->norm) >> GrColorInfo->shift[1]) &  \
(GrColorInfo->mask[1])                                                 \
)
#define GrRGBcolorBlue(c) (                                                   \
(int)(((GrColor)(c) << GrColorInfo->norm) >> GrColorInfo->shift[2]) &  \
(GrColorInfo->mask[2])                                                 \
)
#define GrAllocColorID(r,g,b) (GrColorInfo->RGBmode ?                         \
GrBuildRGBcolorR(r,g,b) :                                              \
GrAllocColor(r,g,b)                                                    \
)
#define GrAllocColor2(hcolor) (GrAllocColor(                                  \
        ((hcolor & 0xff0000) >> 16),                                          \
        ((hcolor & 0x00ff00) >> 8),                                           \
        (hcolor & 0x0000ff))                                                  \
)
#define GrAllocColor2ID(hcolor) (GrAllocColorID(                              \
        ((hcolor & 0xff0000) >> 16),                                          \
        ((hcolor & 0x00ff00) >> 8),                                           \
        (hcolor & 0x0000ff))                                                  \
)
#define GrQueryColorID(c,r,g,b) do {                                          \
if(GrColorInfo->RGBmode) {                                             \
*(r) = GrRGBcolorRed(c);                                               \
*(g) = GrRGBcolorGreen(c);                                             \
*(b) = GrRGBcolorBlue(c);                                              \
break;                                                                 \
}                                                                      \
if(((GrColor)(c) < GrColorInfo->ncolors) &&                            \
   (GrColorInfo->ctable[(GrColor)(c)].defined)) {                      \
*(r) = GrColorInfo->ctable[(GrColor)(c)].r;                            \
*(g) = GrColorInfo->ctable[(GrColor)(c)].g;                            \
*(b) = GrColorInfo->ctable[(GrColor)(c)].b;                            \
break;                                                                 \
}                                                                      \
*(r) = *(g) = *(b) = 0;                                                \
} while(0)
#define GrQueryColor2ID(c,hcolor) do {                                         \
if(GrColorInfo->RGBmode) {                                             \
        *(hcolor) = GrRGBcolorRed(c) << 16;                                    \
        *(hcolor) |= GrRGBcolorGreen(c) << 8;                                  \
        *(hcolor) |= GrRGBcolorBlue(c);                                        \
break;                                                                 \
}                                                                      \
if(((GrColor)(c) < GrColorInfo->ncolors) &&                            \
   (GrColorInfo->ctable[(GrColor)(c)].defined)) {                      \
        *(hcolor) = GrColorInfo->ctable[(GrColor)(c)].r;                       \
        *(hcolor) = GrColorInfo->ctable[(GrColor)(c)].g;                       \
        *(hcolor) = GrColorInfo->ctable[(GrColor)(c)].b;                       \
break;                                                                 \
}                                                                      \
        *(hcolor) = 0;                                                         \
} while(0)
#endif  /* GRX_SKIP_INLINES */

/*
 * color table (for primitives using several colors):
 *   it is an array of colors with the first element being
 *   the number of colors in the table
 */
typedef GrColor *GrColorTableP;

#define GR_CTABLE_SIZE(table) (                                                \
(table) ? (unsigned int)((table)[0]) : 0U                              \
)
#define GR_CTABLE_COLOR(table,index) (                                         \
((unsigned)(index) < GR_CTABLE_SIZE(table)) ?                          \
(table)[((unsigned)(index)) + 1] :                                     \
GrNOCOLOR                                                              \
)
#define GR_CTABLE_ALLOCSIZE(ncolors)    ((ncolors) + 1)

/* ================================================================== */
/*                       GRAPHICS PRIMITIVES                          */
/* ================================================================== */

#ifdef  __TURBOC__
/* this is for GRX compiled with SMALL_STACK: */
#define GR_MAX_POLYGON_POINTS   (8192)
#define GR_MAX_ELLIPSE_POINTS   (1024 + 5)
/* old values without SMALL_STACK: */
/* #define GR_MAX_POLYGON_POINTS   (512) */
/* #define GR_MAX_ELLIPSE_POINTS   (256 + 5) */
#else
#define GR_MAX_POLYGON_POINTS   (1000000)
#define GR_MAX_ELLIPSE_POINTS   (1024 + 5)
#endif
#define GR_MAX_ANGLE_VALUE      (3600)
#define GR_ARC_STYLE_OPEN       0
#define GR_ARC_STYLE_CLOSE1     1
#define GR_ARC_STYLE_CLOSE2     2

typedef struct {                        /* framed box colors */
GrColor fbx_intcolor;
GrColor fbx_topcolor;
GrColor fbx_rightcolor;
GrColor fbx_bottomcolor;
GrColor fbx_leftcolor;
} GrFBoxColors;

void GrClearScreen(GrColor bg);
void GrClearContext(GrColor bg);
void GrClearClipBox(GrColor bg);
void GrPlot(int x,int y,GrColor c);
void GrLine(int x1,int y1,int x2,int y2,GrColor c);
void GrHLine(int x1,int x2,int y,GrColor c);
void GrVLine(int x,int y1,int y2,GrColor c);
void GrBox(int x1,int y1,int x2,int y2,GrColor c);
void GrFilledBox(int x1,int y1,int x2,int y2,GrColor c);
void GrFramedBox(int x1,int y1,int x2,int y2,int wdt,const GrFBoxColors *c);
int  GrGenerateEllipse(int xc,int yc,int xa,int ya,int points[GR_MAX_ELLIPSE_POINTS][2
int  GrGenerateEllipseArc(int xc,int yc,int xa,int ya,int start,int end,int points[GR_
void GrLastArcCoords(int *xs,int *ys,int *xe,int *ye,int *xc,int *yc);
void GrCircle(int xc,int yc,int r,GrColor c);
void GrEllipse(int xc,int yc,int xa,int ya,GrColor c);
void GrCircleArc(int xc,int yc,int r,int start,int end,int style,GrColor c);
void GrEllipseArc(int xc,int yc,int xa,int ya,int start,int end,int style,GrColor c);
void GrFilledCircle(int xc,int yc,int r,GrColor c);
void GrFilledEllipse(int xc,int yc,int xa,int ya,GrColor c);
void GrFilledCircleArc(int xc,int yc,int r,int start,int end,int style,GrColor c);
void GrFilledEllipseArc(int xc,int yc,int xa,int ya,int start,int end,int style,GrColo
void GrPolyLine(int numpts,int points[][2],GrColor c);
void GrPolygon(int numpts,int points[][2],GrColor c);
void GrFilledConvexPolygon(int numpts,int points[][2],GrColor c);
void GrFilledPolygon(int numpts,int points[][2],GrColor c);
void GrBitBlt(GrContext *dst,int x,int y,GrContext *src,int x1,int y1,int x2,int y2,Gr
void GrBitBlt1bpp(GrContext *dst,int dx,int dy,GrContext *src,int x1,int y1,int x2,int
void GrFloodFill(int x, int y, GrColor border, GrColor c);
```

```
GrColor GrPixel(int x,int y);
GrColor GrPixelC(GrContext *c,int x,int y);

const GrColor *GrGetScanline(int x1,int x2,int yy);
const GrColor *GrGetScanlineC(GrContext *ctx,int x1,int x2,int yy);
/* Input   ctx: source context, if NULL the current context is used */
/*         x1 : first x coordinate read                             */
/*         x2 : last  x coordinate read                             */
/*         yy : y coordinate                                        */
/* Output  NULL     : error / no data (clipping occured)            */
/*         else                                                     */
/*           p[0..w]: pixel values read                             */
/*                      (w = |x2-y1|)                               */
/*           Output data is valid until next GRX call !             */

void GrPutScanline(int x1,int x2,int yy,const GrColor *c, GrColor op);
/* Input   x1 : first x coordinate to be set                        */
/*         x2 : last  x coordinate to be set                        */
/*         yy : y coordinate                                        */
/*         c  : c[0..(|x2-x1|] hold the pixel data                  */
/*         op : Operation (GrWRITE/GrXOR/GrOR/GrAND/GrIMAGE)        */
/*                                                                  */
/* Note    c[..] data must fit GrCVALUEMASK otherwise the results   */
/*         are implementation dependend.                            */
/*         => You can't supply operation code with the pixel data!  */


#ifndef GRX_SKIP_INLINES
#define GrGetScanline(x1,x2,yy) \
GrGetScanlineC(NULL,(x1),(x2),(yy))
#endif

/* ================================================================== */
/*                  NON CLIPPING DRAWING PRIMITIVES                   */
/* ================================================================== */

void GrPlotNC(int x,int y,GrColor c);
void GrLineNC(int x1,int y1,int x2,int y2,GrColor c);
void GrHLineNC(int x1,int x2,int y,GrColor c);
void GrVLineNC(int x,int y1,int y2,GrColor c);
void GrBoxNC(int x1,int y1,int x2,int y2,GrColor c);
void GrFilledBoxNC(int x1,int y1,int x2,int y2,GrColor c);
void GrFramedBoxNC(int x1,int y1,int x2,int y2,int wdt,const GrFBoxCol-
ors *c);
void GrBitBltNC(GrContext *dst,int x,int y,GrContext *src,int x1,int y1,int x2,int y2,

GrColor GrPixelNC(int x,int y);
GrColor GrPixelCNC(GrContext *c,int x,int y);

#ifndef GRX_SKIP_INLINES
#define GrPlotNC(x,y,c) (                                                      \
(*GrCurrentFrameDriver()->drawpixel)(                                  \
((x) + GrCurrentContext()->gc_xoffset),                                \
((y) + GrCurrentContext()->gc_yoffset),                                \
((c))                                                                  \
)                                                                      \
)
#define GrPixelNC(x,y) (                                                       \
(*GrCurrentFrameDriver()->readpixel)(                                  \
(GrFrame *)(&GrCurrentContext()->gc_frame),                            \
((x) + GrCurrentContext()->gc_xoffset),                                \
((y) + GrCurrentContext()->gc_yoffset)                                 \
)                                                                      \
)
#define GrPixelCNC(c,x,y) (                                                    \
(*(c)->gc_driver->readpixel)(                                          \
(&(c)->gc_frame),                                                      \
((x) + (c)->gc_xoffset),                                               \
((y) + (c)->gc_yoffset)                                                \
)                                                                      \
)
#endif  /* GRX_SKIP_INLINES */

/* ================================================================== */
/*                   FONTS AND TEXT PRIMITIVES                        */
/* ================================================================== */

/*
 * text drawing directions
 */
#define GR_TEXT_RIGHT           0       /* normal */
#define GR_TEXT_DOWN            1       /* downward */
#define GR_TEXT_LEFT            2       /* upside down, right to left */
#define GR_TEXT_UP              3       /* upward */
#define GR_TEXT_DEFAULT         GR_TEXT_RIGHT
#define GR_TEXT_IS_VERTICAL(d)  ((d) & 1)

/*
 * text alignment options
 */
#define GR_ALIGN_LEFT           0       /* X only */
#define GR_ALIGN_TOP            0       /* Y only */
#define GR_ALIGN_CENTER         1       /* X, Y   */
#define GR_ALIGN_RIGHT          2       /* X only */
```

```
#define GR_ALIGN_BOTTOM         2       /* Y only */
#define GR_ALIGN_BASELINE       3       /* Y only */
#define GR_ALIGN_DEFAULT        GR_ALIGN_LEFT

/*
 * character types in text strings
 */
#define GR_BYTE_TEXT            0       /* one byte per character */
#define GR_WORD_TEXT            1       /* two bytes per character */
#define GR_ATTR_TEXT            2       /* chr w/ PC style attribute byte */

/*
 * macros to access components of various string/character types
 */
#define GR_TEXTCHR_SIZE(ty)     (((ty) == GR_BYTE_TEXT) ? sizeof(char) : sizeof(short)
#define GR_TEXTCHR_CODE(ch,ty)  (((ty) == GR_WORD_TEXT) ? (unsigned short)(ch) : (un-
signed char)(ch))
#define GR_TEXTCHR_ATTR(ch,ty)  (((ty) == GR_ATTR_TEXT) ? ((unsigned short)(ch) >> 8)
#define GR_TEXTSTR_CODE(pt,ty)  (((ty) == GR_WORD_TEXT) ? ((unsigned short *)(pt))[0]
signed char *)(pt))[0])
#define GR_TEXTSTR_ATTR(pt,ty)  (((ty) == GR_ATTR_TEXT) ? ((unsigned char *)(pt))[1] :

/*
 * text attribute macros for the GR_ATTR_TEXT type
 * _GR_textattrintensevideo drives if the eighth bit is used for
 * underline (false, default) or more background colors (true)
 */
extern int _GR_textattrintensevideo;

#define GR_BUILD_ATTR(fg,bg,ul) (_GR_textattrintensevideo ? \
                                (((fg) & 15) | (((bg) & 15) << 4)) \
                                : \
                                (((fg) & 15) | (((bg) & 7) << 4) | ((ul) ? 128 : 0)) \
                                )
#define GR_ATTR_FGCOLOR(attr)   (((attr)     ) &  15)
#define GR_ATTR_BGCOLOR(attr)   (_GR_textattrintensevideo ? \
                                (((attr) >> 4) &  15) \
                                : \
                                (((attr) >> 4) &   7) \
                                )
#define GR_ATTR_UNDERLINE(attr) (_GR_textattrintensevideo ? \
                                (0) \
                                : \
                                (((attr)     ) & 128) \
                                )

/*
```

```
 * OR this to the foreground color value for underlined text when
 * using GR_BYTE_TEXT or GR_WORD_TEXT modes.
 */
#define GR_UNDERLINE_TEXT       (GrXOR << 4)

/*
 * Font conversion flags for 'GrLoadConvertedFont'. OR them as desired.
 */
#define GR_FONTCVT_NONE         0       /* no conversion */
#define GR_FONTCVT_SKIPCHARS    1       /* load only selected characters */
#define GR_FONTCVT_RESIZE       2       /* resize the font */
#define GR_FONTCVT_ITALICIZE    4       /* tilt font for "italic" look */
#define GR_FONTCVT_BOLDIFY      8       /* make a "bold"(er) font  */
#define GR_FONTCVT_FIXIFY       16      /* convert prop. font to fixed wdt */
#define GR_FONTCVT_PROPORTION   32      /* convert fixed font to prop. wdt */

/*
 * font structures
 */
typedef struct _GR_fontHeader {         /* font descriptor */
char    *name;                      /* font name */
char    *family;                    /* font family name */
char     proportional;              /* characters have varying width */
char     scalable;                  /* derived from a scalable font */
char     preloaded;                 /* set when linked into program */
char     modified;                  /* "tweaked" font (resized, etc..) */
unsigned int  width;                /* width (proportional=>average) */
unsigned int  height;               /* font height */
unsigned int  baseline;             /* baseline pixel pos (from top) */
unsigned int  ulpos;                /* underline pixel pos (from top) */
unsigned int  ulheight;             /* underline width */
unsigned int  minchar;              /* lowest character code in font */
unsigned int  numchars;             /* number of characters in font */
} GrFontHeader;

typedef struct _GR_fontChrInfo {        /* character descriptor */
unsigned int  width;                /* width of this character */
unsigned int  offset;               /* offset from start of bitmap */
} GrFontChrInfo;

typedef struct _GR_font {               /* the complete font */
struct  _GR_fontHeader  h;          /* the font info structure */
char     far *bitmap;               /* character bitmap array */
char     far *auxmap;               /* map for rotated & underline chrs */
unsigned int  minwidth;             /* width of narrowest character */
unsigned int  maxwidth;             /* width of widest character */
unsigned int  auxsize;              /* allocated size of auxiliary map */
unsigned int  auxnext;              /* next free byte in auxiliary map */
unsigned int  far      *auxoffs[7]; /* offsets to completed aux chars */
struct  _GR_fontChrInfo chrinfo[1]; /* character info (not act. size) */
} GrFont;

extern  GrFont          GrFont_PC6x8;
extern  GrFont          GrFont_PC8x8;
extern  GrFont          GrFont_PC8x14;
extern  GrFont          GrFont_PC8x16;
#define GrDefaultFont   GrFont_PC8x14

GrFont *GrLoadFont(char *name);
GrFont *GrLoadConvertedFont(char *name,int cvt,int w,int h,int minch,int maxch);
GrFont *GrBuildConvertedFont(const GrFont *from,int cvt,int w,int h,int minch,int maxc

void GrUnloadFont(GrFont *font);
void GrDumpFont(const GrFont *f,char *CsymbolName,char *fileName);
void GrDumpFnaFont(const GrFont *f, char *fileName);
void GrSetFontPath(char *path_list);

int  GrFontCharPresent(const GrFont *font,int chr);
int  GrFontCharWidth(const GrFont *font,int chr);
int  GrFontCharHeight(const GrFont *font,int chr);
int  GrFontCharBmpRowSize(const GrFont *font,int chr);
int  GrFontCharBitmapSize(const GrFont *font,int chr);
int  GrFontStringWidth(const GrFont *font,void *text,int len,int type);
int  GrFontStringHeight(const GrFont *font,void *text,int len,int type);
int  GrProportionalTextWidth(const GrFont *font,const void *text,int len,int type);

char far *GrBuildAuxiliaryBitmap(GrFont *font,int chr,int dir,int ul);
char far *GrFontCharBitmap(const GrFont *font,int chr);
char far *GrFontCharAuxBmp(GrFont *font,int chr,int dir,int ul);

typedef union _GR_textColor {           /* text color union */
GrColor       v;                    /* color value for "direct" text */
GrColorTableP p;                    /* color table for attribute text */
} GrTextColor;

typedef struct _GR_textOption {         /* text drawing option structure */
struct  _GR_font  *txo_font;        /* font to be used */
union  _GR_textColor txo_fgcolor;   /* foreground color */
union  _GR_textColor txo_bgcolor;   /* background color */
char    txo_chrtype;                /* character type (see above) */
char    txo_direct;                 /* direction (see above) */
char    txo_xalign;                 /* X alignment (see above) */
char    txo_yalign;                 /* Y alignment (see above) */
} GrTextOption;
```

```
typedef struct {                        /* fixed font text window desc. */
struct _GR_font     *txr_font;      /* font to be used */
union  _GR_textColor txr_fgcolor;   /* foreground color */
union  _GR_textColor txr_bgcolor;   /* background color */
void   *txr_buffer;                 /* pointer to text buffer */
void   *txr_backup;                 /* optional backup buffer */
int     txr_width;                  /* width of area in chars */
int     txr_height;                 /* height of area in chars */
int     txr_lineoffset;             /* offset in buffer(s) between rows */
int     txr_xpos;                   /* upper left corner X coordinate */
int     txr_ypos;                   /* upper left corner Y coordinate */
char    txr_chrtype;                /* character type (see above) */
} GrTextRegion;

int  GrCharWidth(int chr,const GrTextOption *opt);
int  GrCharHeight(int chr,const GrTextOption *opt);
void GrCharSize(int chr,const GrTextOption *opt,int *w,int *h);
int  GrStringWidth(void *text,int length,const GrTextOption *opt);
int  GrStringHeight(void *text,int length,const GrTextOption *opt);
void GrStringSize(void *text,int length,const GrTextOption *opt,int *w,int *h);

void GrDrawChar(int chr,int x,int y,const GrTextOption *opt);
void GrDrawString(void *text,int length,int x,int y,const GrTextOption *opt);
void GrTextXY(int x,int y,char *text,GrColor fg,GrColor bg);

void GrDumpChar(int chr,int col,int row,const GrTextRegion *r);
void GrDumpText(int col,int row,int wdt,int hgt,const GrTextRegion *r);
void GrDumpTextRegion(const GrTextRegion *r);

#ifndef GRX_SKIP_INLINES
#define GrFontCharPresent(f,ch) (                                              \
((unsigned int)(ch) - (f)->h.minchar) < (f)->h.numchars                \
)
#define GrFontCharWidth(f,ch) (                                                \
GrFontCharPresent(f,ch) ?                                              \
(int)(f)->chrinfo[(unsigned int)(ch) - (f)->h.minchar].width :         \
(f)->h.width                                                           \
)
#define GrFontCharHeight(f,ch) (                                               \
(f)->h.height                                                          \
)
#define GrFontCharBmpRowSize(f,ch) (                                           \
GrFontCharPresent(f,ch) ?                                              \
(((f)->chrinfo[(unsigned int)(ch) - (f)->h.minchar].width + 7) >> 3) : \
0                                                                      \
)
```

```
#define GrFontCharBitmapSize(f,ch) (                                            \
GrFontCharBmpRowSize(f,ch) * (f)->h.height                             \
)
#define GrFontStringWidth(f,t,l,tp) (                                           \
(f)->h.proportional ?                                                  \
GrProportionalTextWidth((f),(t),(l),(tp)) :                            \
(f)->h.width * (l)                                                     \
)
#define GrFontStringHeight(f,t,l,tp) (                                          \
(f)->h.height                                                          \
)
#define GrFontCharBitmap(f,ch) (                                                \
GrFontCharPresent(f,ch) ?                                              \
&(f)->bitmap[(f)->chrinfo[(unsigned int)(ch) - (f)->h.minchar].offset]:\
(char far *)0                                                          \
)
#define GrFontCharAuxBmp(f,ch,dir,ul) (                                         \
(((dir) == GR_TEXT_DEFAULT) && !(ul)) ?                                \
GrFontCharBitmap(f,ch) :                                               \
GrBuildAuxiliaryBitmap((f),(ch),(dir),(ul))                            \
)
#define GrCharWidth(c,o) (                                                      \
GR_TEXT_IS_VERTICAL((o)->txo_direct) ?                                 \
GrFontCharHeight((o)->txo_font,GR_TEXTCHR_CODE(c,(o)->txo_chrtype)) :  \
GrFontCharWidth( (o)->txo_font,GR_TEXTCHR_CODE(c,(o)->txo_chrtype))    \
)
#define GrCharHeight(c,o) (                                                     \
GR_TEXT_IS_VERTICAL((o)->txo_direct) ?                                 \
GrFontCharWidth( (o)->txo_font,GR_TEXTCHR_CODE(c,(o)->txo_chrtype)) :  \
GrFontCharHeight((o)->txo_font,GR_TEXTCHR_CODE(c,(o)->txo_chrtype))    \
)
#define GrCharSize(c,o,wp,hp) do {                                              \
*(wp) = GrCharHeight(c,o);                                             \
*(hp) = GrCharWidth( c,o);                                             \
} while(0)
#define GrStringWidth(t,l,o) (                                                  \
GR_TEXT_IS_VERTICAL((o)->txo_direct) ?                                 \
GrFontStringHeight((o)->txo_font,(t),(l),(o)->txo_chrtype) :           \
GrFontStringWidth( (o)->txo_font,(t),(l),(o)->txo_chrtype)             \
)
#define GrStringHeight(t,l,o) (                                                 \
GR_TEXT_IS_VERTICAL((o)->txo_direct) ?                                 \
GrFontStringWidth( (o)->txo_font,(t),(l),(o)->txo_chrtype) :           \
GrFontStringHeight((o)->txo_font,(t),(l),(o)->txo_chrtype)             \
)
#define GrStringSize(t,l,o,wp,hp) do {                                          \
*(wp) = GrStringWidth( t,l,o);                                         \
*(hp) = GrStringHeight(t,l,o);                                          \
} while(0)
#endif /* GRX_SKIP_INLINES */

/* ================================================================== */
/*             THICK AND DASHED LINE DRAWING PRIMITIVES               */
/* ================================================================== */

/*
 * custom line option structure
 *   zero or one dash pattern length means the line is continuous
 *   the dash pattern always begins with a drawn section
 */
typedef struct {
GrColor lno_color;                  /* color used to draw line */
int     lno_width;                  /* width of the line */
int     lno_pattlen;                /* length of the dash pattern */
unsigned char *lno_dashpat;         /* draw/nodraw pattern */
} GrLineOption;

void GrCustomLine(int x1,int y1,int x2,int y2,const GrLineOption *o);
void GrCustomBox(int x1,int y1,int x2,int y2,const GrLineOption *o);
void GrCustomCircle(int xc,int yc,int r,const GrLineOption *o);
void GrCustomEllipse(int xc,int yc,int xa,int ya,const GrLineOption *o);
void GrCustomCircleArc(int xc,int yc,int r,int start,int end,int style,const Gr-
LineOption *o);
void GrCustomEllipseArc(int xc,int yc,int xa,int ya,int start,int end,int style,const
LineOption *o);
void GrCustomPolyLine(int numpts,int points[][2],const GrLineOption *o);
void GrCustomPolygon(int numpts,int points[][2],const GrLineOption *o);

/* ================================================================== */
/*              PATTERNED DRAWING AND FILLING PRIMITIVES              */
/* ================================================================== */

/*
 * BITMAP: a mode independent way to specify a fill pattern of two
 *   colors. It is always 8 pixels wide (1 byte per scan line), its
 *   height is user-defined. SET THE TYPE FLAG TO ZERO!!!
 */
typedef struct _GR_bitmap {
int     bmp_ispixmap;               /* type flag for pattern union */
int     bmp_height;                 /* bitmap height */
char   *bmp_data;                   /* pointer to the bit pattern */
GrColor bmp_fgcolor;                /* foreground color for fill */
GrColor bmp_bgcolor;                /* background color for fill */
int     bmp_memflags;               /* set if dynamically allocated */
```

```
} GrBitmap;

/*
 * PIXMAP: a fill pattern stored in a layout identical to the video RAM
 *   for filling using 'bitblt'-s. It is mode dependent, typically one
 *   of the library functions is used to build it. KEEP THE TYPE FLAG
 *   NONZERO!!!
 */
typedef struct _GR_pixmap {
int     pxp_ispixmap;               /* type flag for pattern union */
int     pxp_width;                  /* pixmap width (in pixels)  */
int     pxp_height;                 /* pixmap height (in pixels) */
GrColor pxp_oper;                   /* bitblt mode (SET, OR, XOR, AND, IM-
AGE) */
struct _GR_frame pxp_source;        /* source context for fill */
} GrPixmap;

/*
 * Fill pattern union -- can either be a bitmap or a pixmap
 */
typedef union _GR_pattern {
int      gp_ispixmap;               /* nonzero for pixmaps */
GrBitmap gp_bitmap;                 /* fill bitmap */
GrPixmap gp_pixmap;                 /* fill pixmap */
} GrPattern;

#define gp_bmp_data                     gp_bitmap.bmp_data
#define gp_bmp_height                   gp_bitmap.bmp_height
#define gp_bmp_fgcolor                  gp_bitmap.bmp_fgcolor
#define gp_bmp_bgcolor                  gp_bitmap.bmp_bgcolor

#define gp_pxp_width                    gp_pixmap.pxp_width
#define gp_pxp_height                   gp_pixmap.pxp_height
#define gp_pxp_oper                     gp_pixmap.pxp_oper
#define gp_pxp_source                   gp_pixmap.pxp_source

/*
 * Draw pattern for line drawings -- specifies both the:
 *   (1) fill pattern, and the
 *   (2) custom line drawing option
 */
typedef struct {
GrPattern     *lnp_pattern;         /* fill pattern */
GrLineOption  *lnp_option;          /* width + dash pattern */
} GrLinePattern;

GrPattern *GrBuildPixmap(const char *pixels,int w,int h,const GrColorTableP colors);
GrPattern *GrBuildPixmapFromBits(const char *bits,int w,int h,GrColor fgc,GrColor bgc)
GrPattern *GrConvertToPixmap(GrContext *src);

void GrDestroyPattern(GrPattern *p);

void GrPatternedLine(int x1,int y1,int x2,int y2,GrLinePattern *lp);
void GrPatternedBox(int x1,int y1,int x2,int y2,GrLinePattern *lp);
void GrPatternedCircle(int xc,int yc,int r,GrLinePattern *lp);
void GrPatternedEllipse(int xc,int yc,int xa,int ya,GrLinePattern *lp);
void GrPatternedCircleArc(int xc,int yc,int r,int start,int end,int style,GrLinePatter
void GrPatternedEllipseArc(int xc,int yc,int xa,int ya,int start,int end,int style,GrL
void GrPatternedPolyLine(int numpts,int points[][2],GrLinePattern *lp);
void GrPatternedPolygon(int numpts,int points[][2],GrLinePattern *lp);

void GrPatternFilledPlot(int x,int y,GrPattern *p);
void GrPatternFilledLine(int x1,int y1,int x2,int y2,GrPattern *p);
void GrPatternFilledBox(int x1,int y1,int x2,int y2,GrPattern *p);
void GrPatternFilledCircle(int xc,int yc,int r,GrPattern *p);
void GrPatternFilledEllipse(int xc,int yc,int xa,int ya,GrPattern *p);
void GrPatternFilledCircleArc(int xc,int yc,int r,int start,int end,int style,GrPatter
void GrPatternFilledEllipseArc(int xc,int yc,int xa,int ya,int start,int end,int style
void GrPatternFilledConvexPolygon(int numpts,int points[][2],GrPattern *p);
void GrPatternFilledPolygon(int numpts,int points[][2],GrPattern *p);
void GrPatternFloodFill(int x, int y, GrColor border, GrPattern *p);

void GrPatternDrawChar(int chr,int x,int y,const GrTextOption *opt,GrPattern *p);
void GrPatternDrawString(void *text,int length,int x,int y,const GrTex-
tOption *opt,GrPattern *p);
void GrPatternDrawStringExt(void *text,int length,int x,int y,const Gr-
TextOption *opt,GrPattern *p);

/* ================================================================== */
/*                       IMAGE MANIPULATION                           */
/* ================================================================== */

/*
 *  by Michal Stencl Copyright (c) 1998 for GRX
 *  <e-mail>    - [stenclpmd@ba.telecom.sk]
 */

#ifndef GrImage
#define GrImage GrPixmap
#endif

/* Flags for GrImageInverse() */

#define GR_IMAGE_INVERSE_LR  0x01  /* inverse left right */
```

```
#define GR_IMAGE_INVERSE_TD  0x02  /* inverse top down */

GrImage *GrImageBuild(const char *pixels,int w,int h,const GrColorTableP colors);
void     GrImageDestroy(GrImage *i);
void     GrImageDisplay(int x,int y, GrImage *i);
void     GrImageDisplayExt(int x1,int y1,int x2,int y2, GrImage *i);
void     GrImageFilledBoxAlign(int xo,int yo,int x1,int y1,int x2,int y2,GrImage *p);
void     GrImageHLineAlign(int xo,int yo,int x,int y,int width,GrImage *p);
void     GrImagePlotAlign(int xo,int yo,int x,int y,GrImage *p);

GrImage *GrImageInverse(GrImage *p,int flag);
GrImage *GrImageStretch(GrImage *p,int nwidth,int nheight);

GrImage *GrImageFromPattern(GrPattern *p);
GrImage *GrImageFromContext(GrContext *c);
GrImage *GrImageBuildUsedAsPattern(const char *pixels,int w,int h,const Gr-
ColorTableP colors);

GrPattern *GrPatternFromImage(GrImage *p);


#ifndef GRX_SKIP_INLINES
#define GrImageFromPattern(p) \
(((p) && (p)->gp_ispixmap) ? (&(p)->gp_pixmap) : NULL)
#define GrImageFromContext(c) \
(GrImage *)GrConvertToPixmap(c)
#define GrPatternFromImage(p) \
(GrPattern *)(p)
#define GrImageBuildUsedAsPattern(pixels,w,h,colors) \
(GrImage *)GrBuildPixmap(pixels,w,h,colors);
#define GrImageDestroy(i)   \
  GrDestroyPattern((GrPattern *)(i));
#endif

/* ================================================================== */
/*               DRAWING IN USER WINDOW COORDINATES                   */
/* ================================================================== */

void GrSetUserWindow(int x1,int y1,int x2,int y2);
void GrGetUserWindow(int *x1,int *y1,int *x2,int *y2);
void GrGetScreenCoord(int *x,int *y);
void GrGetUserCoord(int *x,int *y);

void GrUsrPlot(int x,int y,GrColor c);
void GrUsrLine(int x1,int y1,int x2,int y2,GrColor c);
void GrUsrHLine(int x1,int x2,int y,GrColor c);
void GrUsrVLine(int x,int y1,int y2,GrColor c);
void GrUsrBox(int x1,int y1,int x2,int y2,GrColor c);
void GrUsrFilledBox(int x1,int y1,int x2,int y2,GrColor c);
void GrUsrFramedBox(int x1,int y1,int x2,int y2,int wdt,GrFBoxColors *c);█
void GrUsrCircle(int xc,int yc,int r,GrColor c);
void GrUsrEllipse(int xc,int yc,int xa,int ya,GrColor c);
void GrUsrCircleArc(int xc,int yc,int r,int start,int end,int style,GrColor c);█
void GrUsrEllipseArc(int xc,int yc,int xa,int ya,int start,int end,int style,GrColor c
void GrUsrFilledCircle(int xc,int yc,int r,GrColor c);
void GrUsrFilledEllipse(int xc,int yc,int xa,int ya,GrColor c);
void GrUsrFilledCircleArc(int xc,int yc,int r,int start,int end,int style,GrColor c);█
void GrUsrFilledEllipseArc(int xc,int yc,int xa,int ya,int start,int end,int style,GrC
void GrUsrPolyLine(int numpts,int points[][2],GrColor c);
void GrUsrPolygon(int numpts,int points[][2],GrColor c);
void GrUsrFilledConvexPolygon(int numpts,int points[][2],GrColor c);
void GrUsrFilledPolygon(int numpts,int points[][2],GrColor c);
void GrUsrFloodFill(int x, int y, GrColor border, GrColor c);

GrColor GrUsrPixel(int x,int y);
GrColor GrUsrPixelC(GrContext *c,int x,int y);

void GrUsrCustomLine(int x1,int y1,int x2,int y2,const GrLineOption *o);
void GrUsrCustomBox(int x1,int y1,int x2,int y2,const GrLineOption *o);
void GrUsrCustomCircle(int xc,int yc,int r,const GrLineOption *o);
void GrUsrCustomEllipse(int xc,int yc,int xa,int ya,const GrLineOption *o);█
void GrUsrCustomCircleArc(int xc,int yc,int r,int start,int end,int style,const Gr-█
LineOption *o);
void GrUsrCustomEllipseArc(int xc,int yc,int xa,int ya,int start,int end,int style,con
LineOption *o);
void GrUsrCustomPolyLine(int numpts,int points[][2],const GrLineOption *o);█
void GrUsrCustomPolygon(int numpts,int points[][2],const GrLineOption *o);█

void GrUsrPatternedLine(int x1,int y1,int x2,int y2,GrLinePattern *lp);
void GrUsrPatternedBox(int x1,int y1,int x2,int y2,GrLinePattern *lp);
void GrUsrPatternedCircle(int xc,int yc,int r,GrLinePattern *lp);
void GrUsrPatternedEllipse(int xc,int yc,int xa,int ya,GrLinePattern *lp);█
void GrUsrPatternedCircleArc(int xc,int yc,int r,int start,int end,int style,GrLinePat
void GrUsrPatternedEllipseArc(int xc,int yc,int xa,int ya,int start,int end,int style,
void GrUsrPatternedPolyLine(int numpts,int points[][2],GrLinePattern *lp);█
void GrUsrPatternedPolygon(int numpts,int points[][2],GrLinePattern *lp);█

void GrUsrPatternFilledPlot(int x,int y,GrPattern *p);
void GrUsrPatternFilledLine(int x1,int y1,int x2,int y2,GrPattern *p);
void GrUsrPatternFilledBox(int x1,int y1,int x2,int y2,GrPattern *p);
void GrUsrPatternFilledCircle(int xc,int yc,int r,GrPattern *p);
void GrUsrPatternFilledEllipse(int xc,int yc,int xa,int ya,GrPattern *p);█
void GrUsrPatternFilledCircleArc(int xc,int yc,int r,int start,int end,int style,GrPat
void GrUsrPatternFilledEllipseArc(int xc,int yc,int xa,int ya,int start,int end,int st
void GrUsrPatternFilledConvexPolygon(int numpts,int points[][2],GrPattern *p);
void GrUsrPatternFilledPolygon(int numpts,int points[][2],GrPattern *p);
void GrUsrPatternFloodFill(int x, int y, GrColor border, GrPattern *p);

void GrUsrDrawChar(int chr,int x,int y,const GrTextOption *opt);
void GrUsrDrawString(char *text,int length,int x,int y,const GrTextOption *opt);
void GrUsrTextXY(int x,int y,char *text,GrColor fg,GrColor bg);

/* =================================================================== */
/*                    GRAPHICS CURSOR UTILITIES                        */
/* =================================================================== */

typedef struct _GR_cursor {
struct _GR_context work;                    /* work areas (4) */
int     xcord,ycord;                        /* cursor position on screen */
int     xsize,ysize;                        /* cursor size */
int     xoffs,yoffs;                        /* LU corner to hot point off-
set */
int     xwork,ywork;                        /* save/work area sizes */
int     xwpos,ywpos;                        /* save/work area position on screen */
int     displayed;                          /* set if displayed */
} GrCursor;

GrCursor *GrBuildCursor(char far *pixels,int pitch,int w,int h,int xo,int yo,const Gr-
ColorTableP c);
void GrDestroyCursor(GrCursor *cursor);
void GrDisplayCursor(GrCursor *cursor);
void GrEraseCursor(GrCursor *cursor);
void GrMoveCursor(GrCursor *cursor,int x,int y);

/* =================================================================== */
/*               MOUSE AND KEYBOARD INPUT UTILITIES                    */
/* =================================================================== */

#define GR_M_MOTION         0x001               /* mouse event flag bits */
#define GR_M_LEFT_DOWN      0x002
#define GR_M_LEFT_UP        0x004
#define GR_M_RIGHT_DOWN     0x008
#define GR_M_RIGHT_UP       0x010
#define GR_M_MIDDLE_DOWN    0x020
#define GR_M_MIDDLE_UP      0x040
#define GR_M_BUTTON_DOWN    (GR_M_LEFT_DOWN | GR_M_MIDDLE_DOWN | GR_M_RIGHT_DOWN)
#define GR_M_BUTTON_UP      (GR_M_LEFT_UP   | GR_M_MIDDLE_UP   | GR_M_RIGHT_UP)
#define GR_M_BUTTON_CHANGE  (GR_M_BUTTON_UP | GR_M_BUTTON_DOWN )

#define GR_M_LEFT           1                   /* mouse button index bits */
#define GR_M_RIGHT          2
```

```
#define GR_M_MIDDLE         4

#define GR_M_KEYPRESS       0x080               /* other event flag bits */
#define GR_M_POLL           0x100
#define GR_M_NOPAINT        0x200
#define GR_SIZE_CHANGED     0x400
#define GR_OS_DRAW_REQUEST  0x800
#define GR_COMMAND          0x1000
#define GR_M_EVENT          (GR_M_MOTION | GR_M_KEYPRESS | GR_M_BUTTON_DOWN | GR_M_BUT

#define GR_KB_RIGHTSHIFT    0x01                /* Keybd states: right shift key de-
pressed */
#define GR_KB_LEFTSHIFT     0x02                /* left shift key depressed */
#define GR_KB_CTRL          0x04                /* CTRL depressed */
#define GR_KB_ALT           0x08                /* ALT depressed */
#define GR_KB_SCROLLOCK     0x10                /* SCROLL LOCK active */
#define GR_KB_NUMLOCK       0x20                /* NUM LOCK active */
#define GR_KB_CAPSLOCK      0x40                /* CAPS LOCK active */
#define GR_KB_INSERT        0x80                /* INSERT state active */
#define GR_KB_SHIFT         (GR_KB_LEFTSHIFT | GR_KB_RIGHTSHIFT)

#define GR_M_CUR_NORMAL     0                   /* MOUSE CURSOR modes: just the cur-
sor */
#define GR_M_CUR_RUBBER     1                   /* rectangular rubber band (XOR-
d to the screen) */
#define GR_M_CUR_LINE       2                   /* line attached to the cur-
sor */
#define GR_M_CUR_BOX        3                   /* rectangular box dragged by the cur-
sor */

#define GR_M_QUEUE_SIZE     128                 /* default queue size */

typedef struct _GR_mouseEvent {                 /* mouse event buffer struc-
ture */
int  flags;                                 /* event type flags (see above) */
int  x,y;                                   /* mouse coordinates */
int  buttons;                               /* mouse button state */
int  key;                                   /* key code from keyboard */
int  kbstat;                                /* keybd status (ALT, CTRL, etc..) */
long dtime;                                 /* time since last event (msec) */
} GrMouseEvent;

/*
 * mouse status information
 */
extern const struct _GR_mouseInfo {
int   (*block)(GrContext*,int,int,int,int); /* mouse block function */
```

```
void  (*unblock)(int flags);                /* mouse unblock function */
void  (*uninit)(void);                      /* mouse cleanupt function */
struct _GR_cursor     *cursor;              /* the mouse cursor */
struct _GR_mouseEvent *queue;               /* queue of pending input events */
int     msstatus;                           /* -1:missing, 0:unknown, 1:present, 2:ini
ted */
int     displayed;                          /* cursor is (generally) drawn */
int     blockflag;                          /* cursor temp. erase/block flag */
int     docheck;                            /* need to check before gr. op. to screen
int     cursmode;                           /* mouse cursor draw mode */
int     x1,y1,x2,y2;                        /* auxiliary params for some cur-
sor draw modes */
GrColor curscolor;                          /* color for some cursor draw modes */
int     owncursor;                          /* auto generated cursor */
int     xpos,ypos;                          /* current mouse position */
int     xmin,xmax;                          /* mouse movement X coordi-
nate limits */
int     ymin,ymax;                          /* mouse movement Y coordi-
nate limits */
int     spmult,spdiv;                       /* mouse cursor speed factors */
int     thresh,accel;                       /* mouse acceleration param-
eters */
int     moved;                              /* mouse cursor movement flag */
int     qsize;                              /* max size of the queue */
int     qlength;                            /* current # of items in the queue */
int     qread;                              /* read pointer for the queue */
int     qwrite;                             /* write pointer for the queue */
} * const GrMouseInfo;

int  GrMouseDetect(void);
void GrMouseEventMode(int dummy);
void GrMouseInit(void);
void GrMouseInitN(int queue_size);
void GrMouseUnInit(void);
void GrMouseSetSpeed(int spmult,int spdiv);
void GrMouseSetAccel(int thresh,int accel);
void GrMouseSetLimits(int x1,int y1,int x2,int y2);
void GrMouseGetLimits(int *x1,int *y1,int *x2,int *y2);
void GrMouseWarp(int x,int y);
void GrMouseEventEnable(int enable_kb,int enable_ms);
void GrMouseGetEvent(int flags,GrMouseEvent *event);

void GrMouseGetEventT(int flags,GrMouseEvent *event,long timout_msecs);
/* Note:
**       event->dtime is only valid if any event occured (event->flags !=0)
**       otherwise it's set as -1.
**       Additionally event timing is now real world time. (X11 && Linux
**       used clock(), user process time, up to 2.28f)
*/

int  GrMousePendingEvent(void);

GrCursor *GrMouseGetCursor(void);
void GrMouseSetCursor(GrCursor *cursor);
void GrMouseSetColors(GrColor fg,GrColor bg);
void GrMouseSetCursorMode(int mode,...);
void GrMouseDisplayCursor(void);
void GrMouseEraseCursor(void);
void GrMouseUpdateCursor(void);
int  GrMouseCursorIsDisplayed(void);

int  GrMouseBlock(GrContext *c,int x1,int y1,int x2,int y2);
void GrMouseUnBlock(int return_value_from_GrMouseBlock);

#if 0
/* !! old style (before grx v2.26) keyboard interface    !!
   !! old functions still linkable but for compatibility !!
   !! across platforms and with future versions of GRX   !!
   !! one use functions from grkeys.h                    !! */
#ifndef __MSDOS__
int  kbhit(void);
int  getch(void);
#endif
#ifndef __DJGPP__
int  getkey(void);
int  getxkey(void);
#endif
int  getkbstat(void);
#endif
/* Why this ???
#ifdef __WATCOMC__
int  getxkey(void);
#endif
*/

#ifndef GRX_SKIP_INLINES
#define GrMouseEventMode(x)          /* nothing! */
#define GrMouseGetCursor()           (GrMouseInfo->cursor)
#define GrMouseCursorIsDisplayed()  (GrMouseInfo->displayed)
#define GrMouseInit()                GrMouseInitN(GR_M_QUEUE_SIZE);
#define GrMouseGetEvent(f,ev)        GrMouseGetEventT((f),(ev),(-1L));
#define GrMousePendingEvent() (                                                 \
GrMouseUpdateCursor(),                                                  \
   (GrMouseInfo->qlength > 0)                                                   \
)
#define GrMouseUnInit() do {                                                   \
if(GrMouseInfo->uninit) {                                              \
(*GrMouseInfo->uninit)();                                              \
}                                                                      \
} while(0)
#define GrMouseGetLimits(x1p,y1p,x2p,y2p) do {                                 \
*(x1p) = GrMouseInfo->xmin; *(y1p) = GrMouseInfo->ymin;                \
*(x2p) = GrMouseInfo->xmax; *(y2p) = GrMouseInfo->ymax;                \
} while(0)
#define GrMouseBlock(c,x1,y1,x2,y2) (                                          \
(((c) ? (const GrContext*)(c) : GrCurrentContext())->gc_onscreen &&    \
 (GrMouseInfo->docheck)) ?                                             \
(*GrMouseInfo->block)((c),(x1),(y1),(x2),(y2)) :                       \
0                                                                      \
)
#define GrMouseUnBlock(f) do {                                                 \
if((f) && GrMouseInfo->displayed) {                                    \
(*GrMouseInfo->unblock)((f));                                          \
}                                                                      \
} while(0)
#endif  /* GRX_SKIP_INLINES */

/* ================================================================== */
/*                           PNM FUNCTIONS                            */
/* ================================================================== */

/*
 *  The PNM formats, grx support load/save of
 *  binaries formats (4,5,6) only
 */

#define PLAINPBMFORMAT 1
#define PLAINPGMFORMAT 2
#define PLAINPPMFORMAT 3
#define PBMFORMAT      4
#define PGMFORMAT      5
#define PPMFORMAT      6

/* The PNM functions */

int GrSaveContextToPbm( GrContext *grc, char *pbmfn, char *docn );
int GrSaveContextToPgm( GrContext *grc, char *pgmfn, char *docn );
int GrSaveContextToPpm( GrContext *grc, char *ppmfn, char *docn );
int GrLoadContextFromPnm( GrContext *grc, char *pnmfn );
int GrQueryPnm( char *pnmfn, int *width, int *height, int *maxval );
int GrLoadContextFromPnmBuffer( GrContext *grc, const char *buffer );
int GrQueryPnmBuffer( const char *buffer, int *width, int *height, int *max-
val );

/* ================================================================== */
/*                          PNG FUNCTIONS                             */
/*  these functions may not be installed or available on all system   */
/* ================================================================== */

int GrPngSupport( void );
int GrSaveContextToPng( GrContext *grc, char *pngfn );
int GrLoadContextFromPng( GrContext *grc, char *pngfn, int use_alpha );
int GrQueryPng( char *pngfn, int *width, int *height );

/* ================================================================== */
/*                         JPEG FUNCTIONS                             */
/*  these functions may not be installed or available on all system   */
/* ================================================================== */

int GrJpegSupport( void );
int GrLoadContextFromJpeg( GrContext *grc, char *jpegfn, int scale );
int GrQueryJpeg( char *jpegfn, int *width, int *height );
int GrSaveContextToJpeg( GrContext *grc, char *jpegfn, int quality );
int GrSaveContextToGrayJpeg( GrContext *grc, char *jpegfn, int quality );

/* ================================================================== */
/*               MISCELLANEOUS UTILITIY FUNCTIONS                     */
/* ================================================================== */

void GrResizeGrayMap(unsigned char far *map,int pitch,int ow,int oh,int nw,int nh);
int  GrMatchString(const char *pattern,const char *strg);
void GrSetWindowTitle(char *title);
void GrSleep(int msec);

/* ================================================================== */
/*                        TIFF ADDON FUNCTIONS                        */
/*  these functions may not be installed or available on all system   */
/* ================================================================== */

/*
** SaveContextToTiff - Dump a context in a TIFF file
**
** Arguments:
**   cxt:   Context to be saved (NULL -> use current context)
**   tiffn: Name of tiff file
**   compr: Compression method (see tiff.h), 0: automatic selection
**   docn:  string saved in the tiff file (DOCUMENTNAME tag)
**
```

```
**  Returns  0 on success
**          -1 on error
**
** requires tifflib by  Sam Leffler (sam@engr.sgi.com)
**        available at  ftp://ftp.sgi.com/graphics/tiff
*/
int SaveContextToTiff(GrContext *cxt, char *tiffn, unsigned compr, char *docn);

#ifdef __cplusplus
}
#endif
#endif  /* whole file */
```

grxkeys.h

```
/**
 ** grxkeys.h ---- platform independent key definitions
 **
 ** Copyright (c) 1997 Hartmut Schirmer
 **
 ** This file is part of the GRX graphics library.
 **
 ** The GRX graphics library is free software; you can redistribute it
 ** and/or modify it under some conditions; see the "copying.grx" file
 ** for details.
 **
 ** This library is distributed in the hope that it will be useful,
 ** but WITHOUT ANY WARRANTY; without even the implied warranty of
 ** MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 **
 **/

#ifndef __GRKEYS_H_INCLUDED__
#define __GRKEYS_H_INCLUDED__

/*
** NOTES - some keys may not be available under all systems
**       - key values will be differ on different systems
*/

#ifndef __GRX20_H_INCLUDED__
#include <grx20.h>
#endif

#ifdef __cplusplus
extern "C" {
#endif
```

```
/* all keycodes should fit into 16 bit unsigned */
typedef unsigned short GrKeyType;

/* no key available */
#define GrKey_NoKey                   0x0000

/* key typed but code outside 1..GrKey_LastDefinedKeycode */
#define GrKey_OutsideValidRange       0x0100

/* standard ASCII key codes */
#define GrKey_Control_A               0x0001
#define GrKey_Control_B               0x0002
#define GrKey_Control_C               0x0003
#define GrKey_Control_D               0x0004
#define GrKey_Control_E               0x0005
#define GrKey_Control_F               0x0006
#define GrKey_Control_G               0x0007
#define GrKey_Control_H               0x0008
#define GrKey_Control_I               0x0009
#define GrKey_Control_J               0x000a
#define GrKey_Control_K               0x000b
#define GrKey_Control_L               0x000c
#define GrKey_Control_M               0x000d
#define GrKey_Control_N               0x000e
#define GrKey_Control_O               0x000f
#define GrKey_Control_P               0x0010
#define GrKey_Control_Q               0x0011
#define GrKey_Control_R               0x0012
#define GrKey_Control_S               0x0013
#define GrKey_Control_T               0x0014
#define GrKey_Control_U               0x0015
#define GrKey_Control_V               0x0016
#define GrKey_Control_W               0x0017
#define GrKey_Control_X               0x0018
#define GrKey_Control_Y               0x0019
#define GrKey_Control_Z               0x001a
#define GrKey_Control_LBracket        0x001b
#define GrKey_Control_BackSlash       0x001c
#define GrKey_Control_RBracket        0x001d
#define GrKey_Control_Caret           0x001e
#define GrKey_Control_Underscore      0x001f
#define GrKey_Space                   0x0020
#define GrKey_ExclamationPoint        0x0021
#define GrKey_DoubleQuote             0x0022
#define GrKey_Hash                    0x0023
#define GrKey_Dollar                  0x0024
```

```
#define GrKey_Percent                0x0025
#define GrKey_Ampersand              0x0026
#define GrKey_Quote                  0x0027
#define GrKey_LParen                 0x0028
#define GrKey_RParen                 0x0029
#define GrKey_Star                   0x002a
#define GrKey_Plus                   0x002b
#define GrKey_Comma                  0x002c
#define GrKey_Dash                   0x002d
#define GrKey_Period                 0x002e
#define GrKey_Slash                  0x002f
#define GrKey_0                      0x0030
#define GrKey_1                      0x0031
#define GrKey_2                      0x0032
#define GrKey_3                      0x0033
#define GrKey_4                      0x0034
#define GrKey_5                      0x0035
#define GrKey_6                      0x0036
#define GrKey_7                      0x0037
#define GrKey_8                      0x0038
#define GrKey_9                      0x0039
#define GrKey_Colon                  0x003a
#define GrKey_SemiColon              0x003b
#define GrKey_LAngle                 0x003c
#define GrKey_Equals                 0x003d
#define GrKey_RAngle                 0x003e
#define GrKey_QuestionMark           0x003f
#define GrKey_At                     0x0040
#define GrKey_A                      0x0041
#define GrKey_B                      0x0042
#define GrKey_C                      0x0043
#define GrKey_D                      0x0044
#define GrKey_E                      0x0045
#define GrKey_F                      0x0046
#define GrKey_G                      0x0047
#define GrKey_H                      0x0048
#define GrKey_I                      0x0049
#define GrKey_J                      0x004a
#define GrKey_K                      0x004b
#define GrKey_L                      0x004c
#define GrKey_M                      0x004d
#define GrKey_N                      0x004e
#define GrKey_O                      0x004f
#define GrKey_P                      0x0050
#define GrKey_Q                      0x0051
#define GrKey_R                      0x0052
#define GrKey_S                      0x0053
```

```
#define GrKey_T                      0x0054
#define GrKey_U                      0x0055
#define GrKey_V                      0x0056
#define GrKey_W                      0x0057
#define GrKey_X                      0x0058
#define GrKey_Y                      0x0059
#define GrKey_Z                      0x005a
#define GrKey_LBracket               0x005b
#define GrKey_BackSlash              0x005c
#define GrKey_RBracket               0x005d
#define GrKey_Caret                  0x005e
#define GrKey_UnderScore             0x005f
#define GrKey_BackQuote              0x0060
#define GrKey_a                      0x0061
#define GrKey_b                      0x0062
#define GrKey_c                      0x0063
#define GrKey_d                      0x0064
#define GrKey_e                      0x0065
#define GrKey_f                      0x0066
#define GrKey_g                      0x0067
#define GrKey_h                      0x0068
#define GrKey_i                      0x0069
#define GrKey_j                      0x006a
#define GrKey_k                      0x006b
#define GrKey_l                      0x006c
#define GrKey_m                      0x006d
#define GrKey_n                      0x006e
#define GrKey_o                      0x006f
#define GrKey_p                      0x0070
#define GrKey_q                      0x0071
#define GrKey_r                      0x0072
#define GrKey_s                      0x0073
#define GrKey_t                      0x0074
#define GrKey_u                      0x0075
#define GrKey_v                      0x0076
#define GrKey_w                      0x0077
#define GrKey_x                      0x0078
#define GrKey_y                      0x0079
#define GrKey_z                      0x007a
#define GrKey_LBrace                 0x007b
#define GrKey_Pipe                   0x007c
#define GrKey_RBrace                 0x007d
#define GrKey_Tilde                  0x007e
#define GrKey_Control_Backspace      0x007f

/* extended key codes as defined in DJGPP */
#define GrKey_Alt_Escape             0x0101
```

```
#define GrKey_Control_At           0x0103
#define GrKey_Alt_Backspace        0x010e
#define GrKey_BackTab              0x010f
#define GrKey_Alt_Q                0x0110
#define GrKey_Alt_W                0x0111
#define GrKey_Alt_E                0x0112
#define GrKey_Alt_R                0x0113
#define GrKey_Alt_T                0x0114
#define GrKey_Alt_Y                0x0115
#define GrKey_Alt_U                0x0116
#define GrKey_Alt_I                0x0117
#define GrKey_Alt_O                0x0118
#define GrKey_Alt_P                0x0119
#define GrKey_Alt_LBracket         0x011a
#define GrKey_Alt_RBracket         0x011b
#define GrKey_Alt_Return           0x011c
#define GrKey_Alt_A                0x011e
#define GrKey_Alt_S                0x011f
#define GrKey_Alt_D                0x0120
#define GrKey_Alt_F                0x0121
#define GrKey_Alt_G                0x0122
#define GrKey_Alt_H                0x0123
#define GrKey_Alt_J                0x0124
#define GrKey_Alt_K                0x0125
#define GrKey_Alt_L                0x0126
#define GrKey_Alt_Semicolon        0x0127
#define GrKey_Alt_Quote            0x0128
#define GrKey_Alt_Backquote        0x0129
#define GrKey_Alt_Backslash        0x012b
#define GrKey_Alt_Z                0x012c
#define GrKey_Alt_X                0x012d
#define GrKey_Alt_C                0x012e
#define GrKey_Alt_V                0x012f
#define GrKey_Alt_B                0x0130
#define GrKey_Alt_N                0x0131
#define GrKey_Alt_M                0x0132
#define GrKey_Alt_Comma            0x0133
#define GrKey_Alt_Period           0x0134
#define GrKey_Alt_Slash            0x0135
#define GrKey_Alt_KPStar           0x0137
#define GrKey_F1                   0x013b
#define GrKey_F2                   0x013c
#define GrKey_F3                   0x013d
#define GrKey_F4                   0x013e
#define GrKey_F5                   0x013f
#define GrKey_F6                   0x0140
#define GrKey_F7                   0x0141
```

```
#define GrKey_F8                    0x0142
#define GrKey_F9                    0x0143
#define GrKey_F10                   0x0144
#define GrKey_Home                  0x0147
#define GrKey_Up                    0x0148
#define GrKey_PageUp                0x0149
#define GrKey_Alt_KPMinus           0x014a
#define GrKey_Left                  0x014b
#define GrKey_Center                0x014c
#define GrKey_Right                 0x014d
#define GrKey_Alt_KPPlus            0x014e
#define GrKey_End                   0x014f
#define GrKey_Down                  0x0150
#define GrKey_PageDown              0x0151
#define GrKey_Insert                0x0152
#define GrKey_Delete                0x0153
#define GrKey_Shift_F1              0x0154
#define GrKey_Shift_F2              0x0155
#define GrKey_Shift_F3              0x0156
#define GrKey_Shift_F4              0x0157
#define GrKey_Shift_F5              0x0158
#define GrKey_Shift_F6              0x0159
#define GrKey_Shift_F7              0x015a
#define GrKey_Shift_F8              0x015b
#define GrKey_Shift_F9              0x015c
#define GrKey_Shift_F10             0x015d
#define GrKey_Control_F1            0x015e
#define GrKey_Control_F2            0x015f
#define GrKey_Control_F3            0x0160
#define GrKey_Control_F4            0x0161
#define GrKey_Control_F5            0x0162
#define GrKey_Control_F6            0x0163
#define GrKey_Control_F7            0x0164
#define GrKey_Control_F8            0x0165
#define GrKey_Control_F9            0x0166
#define GrKey_Control_F10           0x0167
#define GrKey_Alt_F1                0x0168
#define GrKey_Alt_F2                0x0169
#define GrKey_Alt_F3                0x016a
#define GrKey_Alt_F4                0x016b
#define GrKey_Alt_F5                0x016c
#define GrKey_Alt_F6                0x016d
#define GrKey_Alt_F7                0x016e
#define GrKey_Alt_F8                0x016f
#define GrKey_Alt_F9                0x0170
#define GrKey_Alt_F10               0x0171
#define GrKey_Control_Print         0x0172
```

```
#define GrKey_Control_Left          0x0173
#define GrKey_Control_Right         0x0174
#define GrKey_Control_End           0x0175
#define GrKey_Control_PageDown      0x0176
#define GrKey_Control_Home          0x0177
#define GrKey_Alt_1                 0x0178
#define GrKey_Alt_2                 0x0179
#define GrKey_Alt_3                 0x017a
#define GrKey_Alt_4                 0x017b
#define GrKey_Alt_5                 0x017c
#define GrKey_Alt_6                 0x017d
#define GrKey_Alt_7                 0x017e
#define GrKey_Alt_8                 0x017f
#define GrKey_Alt_9                 0x0180
#define GrKey_Alt_0                 0x0181
#define GrKey_Alt_Dash              0x0182
#define GrKey_Alt_Equals            0x0183
#define GrKey_Control_PageUp        0x0184
#define GrKey_F11                   0x0185
#define GrKey_F12                   0x0186
#define GrKey_Shift_F11             0x0187
#define GrKey_Shift_F12             0x0188
#define GrKey_Control_F11           0x0189
#define GrKey_Control_F12           0x018a
#define GrKey_Alt_F11               0x018b
#define GrKey_Alt_F12               0x018c
#define GrKey_Control_Up            0x018d
#define GrKey_Control_KPDash        0x018e
#define GrKey_Control_Center        0x018f
#define GrKey_Control_KPPlus        0x0190
#define GrKey_Control_Down          0x0191
#define GrKey_Control_Insert        0x0192
#define GrKey_Control_Delete        0x0193
#define GrKey_Control_Tab           0x0194
#define GrKey_Control_KPSlash       0x0195
#define GrKey_Control_KPStar        0x0196
#define GrKey_Alt_KPSlash           0x01a4
#define GrKey_Alt_Tab               0x01a5
#define GrKey_Alt_Enter             0x01a6

/* some additional codes not in DJGPP */
#define GrKey_Alt_LAngle            0x01b0
#define GrKey_Alt_RAngle            0x01b1
#define GrKey_Alt_At                0x01b2
#define GrKey_Alt_LBrace            0x01b3
#define GrKey_Alt_Pipe              0x01b4
#define GrKey_Alt_RBrace            0x01b5
```

```
#define GrKey_Print                 0x01b6
#define GrKey_Shift_Insert          0x01b7
#define GrKey_Shift_Home            0x01b8
#define GrKey_Shift_End             0x01b9
#define GrKey_Shift_PageUp          0x01ba
#define GrKey_Shift_PageDown        0x01bb
#define GrKey_Alt_Up                0x01bc
#define GrKey_Alt_Left              0x01bd
#define GrKey_Alt_Center            0x01be
#define GrKey_Alt_Right             0x01c0
#define GrKey_Alt_Down              0x01c1
#define GrKey_Alt_Insert            0x01c2
#define GrKey_Alt_Delete            0x01c3
#define GrKey_Alt_Home              0x01c4
#define GrKey_Alt_End               0x01c5
#define GrKey_Alt_PageUp            0x01c6
#define GrKey_Alt_PageDown          0x01c7
#define GrKey_Shift_Up              0x01c8
#define GrKey_Shift_Down            0x01c9
#define GrKey_Shift_Right           0x01ca
#define GrKey_Shift_Left            0x01cb

/* this may be usefull for table allocation ... */
#define GrKey_LastDefinedKeycode   GrKey_Shift_Left

/* some well known synomyms */
#define GrKey_BackSpace             GrKey_Control_H
#define GrKey_Tab                   GrKey_Control_I
#define GrKey_LineFeed              GrKey_Control_J
#define GrKey_Escape                GrKey_Control_LBracket
#define GrKey_Return                GrKey_Control_M

/*
** new functions to replace the old style
**   kbhit / getch / getkey / getxkey / getkbstat
** keyboard interface
*/
extern int GrKeyPressed(void);
extern GrKeyType GrKeyRead(void);
extern int GrKeyStat(void);

/* some compatibility interface here ?? */
/* eg., #define kbhit() GrKeyPressed() ?? */

#ifdef __cplusplus
}
#endif
```

```
#endif /* whole file */
```

Table of Contents

GRX2 User's Manual 1

Credits 1
Hello world 1
Data types and function declarations 2
Setting the graphics driver 2
Setting video modes 3
Graphics contexts 5
Context use 8
Color management 9
Portable use of a few colors 11
Graphics primitives 11
Non-clipping graphics primitives 14
Customized line drawing 15
Pattern filled graphics primitives 15
Patterned line drawing 18
Image manipulation 18
Text drawing 19
Drawing in user coordinates 24
Graphics cursors 26
Keyboard input 27
Mouse event handling 28
Writing/reading PNM graphics files 31
Writing/reading PNG graphics files 32
Writing/reading PNG graphics files 33
Miscellaneous functions 34
BGI interface 35
Test examples 35
arctest.c 35
bb1test.c 37
blittest.c 38
circtest.c 45
cliptest.c 47
colorops.c 48
curstest.c 51
demogrx.c 53
fontdemo.c 64
fonttest.c 69
fswwtest.c 73
gfaz.c 75
imgtest.c 83
jpgtest.c 85
keys.c 88
life.c 98

linetest.c 100
memtest.c 102
mousetst.c 103
pcirctst.c 105
pngtest.c 108
pnmtest.c 110
polytest.c 112
rgbtest.c 116
sbctest.c 117
scroltst.c 121
speedtst.c 122
textpatt.c 142
winclip.c 143
wintest.c 144
drawing.h 146
gfaz.h 146
rand.h 149
test.h 150
arctest.dat 152
polytest.dat 152
Includes 154
grx20.h 154
grxkeys.h 191