```
/*
 * $Id: gpr.trm%v 3.50 1993/07/09 05:35:24 woo Exp $
 *
 */

/* GNUPLOT - gpr.trm */
/*
 * Copyright (C) 1990 - 1993
 *
 * Permission to use, copy, and distribute this software and its
 * documentation for any purpose with or without fee is hereby granted,
 * provided that the above copyright notice appear in all copies and
 * that both that copyright notice and this permission notice appear
 * in supporting documentation.
 *
 * Permission to modify the software is granted, but not the right to
 * distribute the modified code.  Modifications are to be distributed
 * as patches to released version.
 *
 * This software  is provided "as is" without express or implied warranty.
 *
 * This file is included by ../term.c.
 *
 * This terminal driver supports:
 *   APOLLO's GPR windowing system
 *
 * AUTHORS
 *  Michael Aramini
 *  Roque D Oliveira , oliveria@caen.engin.umich.edu
 *
 * send your comments or suggestions to (info-gnuplot@dartmouth.edu).
 *
 */
#include <apollo/base.h>
#include <apollo/error.h>
#include <apollo/gpr.h>
#include <apollo/pad.h>
/* landscape window */
/*
#define GPR_XMAX 720
#define GPR_YMAX 450
*/
/* portrait window */
#define GPR_XMAX 585
#define GPR_YMAX 735

#define GPR_XLAST (GPR_XMAX - 1)
```

```
#define GPR_YLAST (GPR_YMAX - 1)

#define GPR_VCHAR 19
#define GPR_HCHAR 10
#define GPR_VTIC (GPR_YMAX/80)
#define GPR_HTIC (GPR_XMAX/80)

gpr_$direction_t  gpr_path=gpr_$right;
int               gpr_ang=0;            /* text angle, 0=horizontal,
1=vertical */
enum JUSTIFY      gpr_justify=LEFT;     /* text is flush left */

static status_$t  status;
unsigned int      Debug    = 0;         /* set it to 1 when debugging
program */


static void check(messagex)
char *messagex;
{
  if (status.all = status_$ok)
  {
    error_$print(status);
    printf("Error occurred while %s.\n", messagex);
  }
}

/* return whether stdout is a DM pad . Called by term.c */
gpr_isa_pad()
{
    pad_$isa(1, &status);
    return (status.all == status_$ok);
}


GPR_init()
{
  gpr_$offset_t dm_bitmap_size;
  gpr_$bitmap_desc_t dm_bitmap_desc;
  pad_$window_desc_t window;
  short font_id;
  stream_$id_t stream_id;
  static gpr_$rgb_plane_t  hi_plane;
  static gpr_$disp_char_t  display_characteristics;
  static float             screen_size_r_width,screen_size_r_height ;
  static short int         disp_len = sizeof(gpr_$disp_char_t);
  static short int         disp_len_returned;
```

```
  /* open a pad to do graphics in */
  window.top    = 0;
  window.left   = 0;
  window.width  = GPR_XMAX + 10; /* 10 accounts for width of window border
*/
  window.height = GPR_YMAX + 35; /* 35 accounts for height of window border
*/
  pad_$create_window("",(short)0,pad_$transcript,
(short)1,window,&stream_id,&status);
  check("pad_$create_window");

/*  pad_$set_full_window(stream_id,(short) 1,&window, &status); */
/*  pad_$set_border (stream_id,(short) 1, true, &status); */
  pad_$set_scale (stream_id,(short) 1,(short) 1, &status);
  pad_$set_auto_close(stream_id, (short) 1, true, &status );


gpr_$inq_disp_characteristics(gpr_$direct,stream_id,disp_len,&display_chara
cteristics,&disp_len_returned,&status);
  check("in gpr_$inq_display_characteristics");
  screen_size_r_width    = (float) display_characteristics.x_window_size;
/*x_window_size in pixels */
  screen_size_r_height   = (float) display_characteristics.y_window_size;
/*y_window_size in pixels */
  hi_plane  = display_characteristics.n_planes - 1;
  if(Debug) printf("width=%f height=%f
\n",screen_size_r_width,screen_size_r_height);

  dm_bitmap_size.x_size = 1280;
  dm_bitmap_size.y_size = 1024;

gpr_$init(gpr_$direct,stream_id,dm_bitmap_size,hi_plane,&dm_bitmap_desc,&st
atus);
  check("in gpr_$init");
/*
  gpr_$set_obscured_opt(gpr_$pop_if_obs, &status);
  check("in gpr_$set_obscured_opt");
*/
  gpr_$set_auto_refresh(true, &status);
  check("in gpr_$set_auto_refresh");

  /* load a font and make it current */
  gpr_$load_font_file("f7x13", 5, &font_id, &status);
  check("in gpr_$load_font_file");
  gpr_$set_text_font(font_id, &status);
```

```
  check("in gpr_$set_text_font");

  /* set up color values */
  gpr_$set_draw_value((gpr_$pixel_value_t)7, &status);  /* white */
  check("in gpr_set_draw_value");
  gpr_$set_text_background_value((gpr_$pixel_value_t)(-1), &status); /*
trans */
  check("in gpr_$set_text_background_value");
  gpr_$set_text_value((gpr_$pixel_value_t)7, &status);  /* white */
  check("in gpr_$set_text_value");
}


GPR_graphics()
{
  gpr_$coordinate_t locx,locy,marker_size;

  (void) gpr_$acquire_display(&status);
  check("in gpr_$acquire display");
  gpr_$clear((gpr_$pixel_value_t)0, &status);  /* black */
  check("in gpr_$clear");

  if(Debug)
   {
    marker_size = (short) 10;

    locx = (short) 5;
    locy = (short) 5;
    gpr_$set_draw_value((gpr_$pixel_value_t)2, &status);  /* white */
    gpr_$move( (locx - marker_size/2) ,locy,&status);
    gpr_$line( (locx + marker_size/2) ,locy,&status);
    gpr_$move( locx, (locy + marker_size/2),&status);
    gpr_$line( locx, (locy - marker_size/2),&status);

    locx = (short) (GPR_XMAX -1 - 5);
    locy = (short) 5;
    gpr_$set_draw_value((gpr_$pixel_value_t)3, &status);  /* white */
    gpr_$move( (locx - marker_size/2) ,locy,&status);
    gpr_$line( (locx + marker_size/2) ,locy,&status);
    gpr_$move( locx, (locy + marker_size/2),&status);
    gpr_$line( locx, (locy - marker_size/2),&status);

    locx = (short) 5;
    locy = (short) (GPR_YMAX -1 - 5);
    gpr_$set_draw_value((gpr_$pixel_value_t)4, &status);  /* white */
    gpr_$move( (locx - marker_size/2) ,locy,&status);
    gpr_$line( (locx + marker_size/2) ,locy,&status);
```

```
    gpr_$move( locx, (locy + marker_size/2),&status);
    gpr_$line( locx, (locy - marker_size/2),&status);

    locx = (short) (GPR_XMAX -1 - 5);
    locy = (short) (GPR_YMAX -1 - 5);
    gpr_$set_draw_value((gpr_$pixel_value_t)5, &status);  /* white */
    gpr_$move( (locx - marker_size/2) ,locy,&status);
    gpr_$line( (locx + marker_size/2) ,locy,&status);
    gpr_$move( locx, (locy + marker_size/2),&status);
    gpr_$line( locx, (locy - marker_size/2),&status);

    gpr_$set_draw_value((gpr_$pixel_value_t)7, &status);  /* white */
    check("in gpr_$set_draw_value");
   } /* end if(Debug) */

}


GPR_text()
{
  gpr_$release_display(&status);
  check("gpr_$release_display");
}


GPRold_linetype(linetype)
int linetype;
{
  static gpr_$line_pattern_t patterns[2+5] = {
      { 0xFFFF },   /* solid              1111111111111111 */
      { 0x3FFF },   /* very long dashed   0011111111111111 */
      { 0xFFFF },   /* solid              1111111111111111 */
      { 0x5555 },   /* dotted             0101010101010101 */
      { 0x3333 },   /* short dashed       0011001100110011 */
      { 0xB5AD },   /* dot short-dashed   1011010110101101 */
      { 0x3FFF }    /* very long dashed   0011111111111111 */
                                             };
  if (linetype >= 5) linetype %= 5;
  gpr_$set_line_pattern((short)1, patterns[linetype+2], (short)16,
&status);
  check("in gpr_$set_line_pattern");
}

GPR_linetype(linetype)
int linetype;
{
  static gpr_$line_pattern_t patterns[2+7] = {
```

```
      { 0xFFFF },   /* solid              1111111111111111 */
      { 0x1111 },   /* long-spaced dotted 0001000100010001 */
      { 0xFFFF },   /* solid              1111111111111111 */
      { 0x5555 },   /* dotted             0101010101010101 */
      { 0x3333 },   /* short  dashed      0011001100110011 */
      { 0x7777 },   /* medium dashed      0111011101110111 */
      { 0x3F3F },   /* long   dashed      0011111100111111 */
      { 0x0F0F },   /* long-spaced dashed 0000111100001111 */
      { 0x5F5F }    /* dot dashed         0101111101011111 */
                                            };
  if (linetype >= 7) linetype %= 7;
  gpr_$set_line_pattern((short)1, patterns[linetype+2], (short)16,
&status);
  check("in gpr_$set_line_pattern");

/*
  gpr_$set_draw_value((gpr_$pixel_value_t)(linetype + 1), &status);
  check("in gpr_$set_draw_value");
*/
}


GPR_move(x, y)
unsigned int x, y;
{
  gpr_$move((short)x, (short)(GPR_YMAX -1 - y), &status);
  check("in gpr_$move");
}


GPR_vector(x, y)
unsigned int x, y;
{
  gpr_$line((short)x, (short)(GPR_YMAX -1 - y), &status);
  check("in gpr_$line");
}


GPR_put_text(x,y,str)
unsigned int x,y;        /* reference point of string */
char str[];              /* the text */
{
  gpr_$coordinate_t xgpr,ygpr;
  gpr_$offset_t     str_size_in_pixels;
  short int         str_len;

  gpr_$coordinate_t locx,locy,marker_size;
```

```
  if(Debug)
   {
    locx = (short) x;
    locy = (short) (GPR_YMAX -1 - y);
    marker_size = (short) 20;
    gpr_$set_draw_value((gpr_$pixel_value_t)1, &status);  /* white */
    gpr_$move( (locx - marker_size/2) ,locy,&status);
    gpr_$line( (locx + marker_size/2) ,locy,&status);
    gpr_$move( locx, (locy + marker_size/2),&status);
    gpr_$line( locx, (locy - marker_size/2),&status);
    gpr_$set_draw_value((gpr_$pixel_value_t)7, &status);  /* white */
   }

  xgpr = (short) x;
  ygpr = (short) (GPR_YMAX -1 - y);
  gpr_$set_text_path(gpr_path, &status);
  check("gpr_$set_text_path");

  str_len = (short) strlen(str);
  gpr_$inq_text_extent(str,str_len,&str_size_in_pixels,&status); /*
Calculate how much space (in pixels) the string requires */
  check("in gpr_$inq_text_extent");

  switch (gpr_justify)
   {
    case LEFT :
        {
         switch (gpr_path)
          {
           case gpr_$up :           /* vertical */
               {
                if(Debug) printf("LEFT and up , str=%s\n",str);
                break;
               }
           case gpr_$right :        /* horizontal */
               {
                ygpr = ygpr + str_size_in_pixels.y_size/2;
                if(Debug) printf("LEFT and right, str=%s \n",str);
                break;
               }
          }
         break;
        }

    case CENTRE :
        {
```

```
       switch (gpr_path)
        {
         case gpr_$up :           /* vertical */
             {
              xgpr = xgpr + str_size_in_pixels.x_size/2;
              ygpr = ygpr + str_size_in_pixels.y_size/2;
              if(Debug) printf("CENTRE and up, str=%s \n",str);
              break;
             }
         case gpr_$right :        /* horizontal */
             {
              xgpr = xgpr - str_size_in_pixels.x_size/2;
              ygpr = ygpr + str_size_in_pixels.y_size/2;
              if(Debug) printf("CENTRE and right, str=%s \n",str);
              break;
             }
        }
       break;
      }
   case RIGHT :
      {
       switch (gpr_path)
        {
         case gpr_$up :           /* vertical */
             {
              ygpr = ygpr + str_size_in_pixels.y_size;
              if(Debug) printf("RIGHT and up, str=%s \n",str);
              break;
             }
         case gpr_$right :        /* horizontal */
             {
              xgpr = xgpr - str_size_in_pixels.x_size;
              ygpr = ygpr + str_size_in_pixels.y_size/2;
              if(Debug) printf("RIGHT and right, str=%s \n",str);
              break;
             }
        }
       break;
      }
   }

  gpr_$move(xgpr,ygpr,&status);
  check("in gpr_$move");
  gpr_$text(str, str_len, &status);
  check("in gpr_$text");
}
```

```
int GPR_text_angle(ang)
int ang;
{
    if (gpr_ang != ang)
      {
        gpr_ang  = ang;
        gpr_path = (gpr_ang == 1 ? gpr_$up : gpr_$right);
      }
    return (TRUE);
}

int GPR_justify_text(mode)
enum JUSTIFY mode;
{
    gpr_justify = mode;
    return (TRUE);
}

GPR_reset()
{
  gpr_$terminate(false, &status);
  check("in gpr_$terminate");
}
```