6 Graphics with GD library

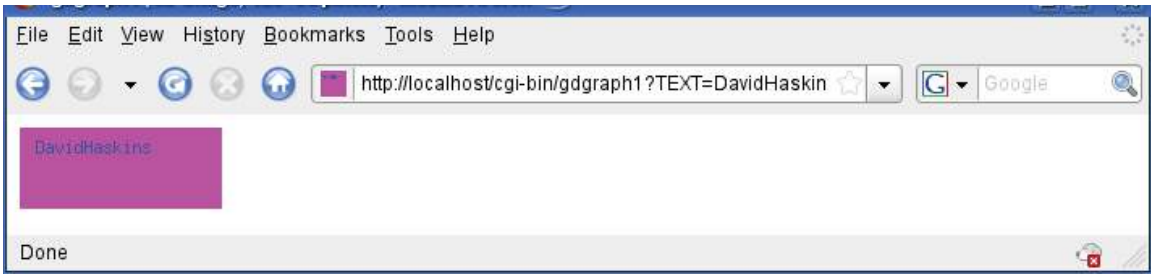
The ability to generate interesting images dynamically to suit the situation is an enjoyable, challenging, and rewarding type of programming. We will be using Thomas Boutell's GD C library which has been around for many years as an open systems project.

6.1 Generating binary content

Here we create an image and print into it the value of the TEXT= part of the QUERY STRING and set our content type to image/gif. The **gdImageGif** function writes the binary image out to **stdout** which is the output stream instead of to a file, so binary image data is sent back to the browser.

```
/*****************************************************************
* C Programming in Linux (c) David Haskins 2008
* chapter6_1.c                                  *
*****************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <gd.h>
#include <gdfonts.h>
#include "c_in_linux.h"

int main(int argc, char *argv[], char *env[])
{
        int text=0,background=0,height=50,width=0;
        char value[255] = "";
        gdImagePtr im_out = NULL;

        decode_value("TEXT=", (char *) &value, 255);
        width = ( strlen(value) * 10 ) + 5;
        im_out = gdImageCreate(width,height);
        background = gdImageColorAllocate(im_out, 255,0,255);
        text  = gdImageColorAllocate(im_out,0,0,255);
        gdImageFilledRectangle(im_out, 0,0, width-1, height-1,  background);
        gdImageString(im_out,gdFontGetSmall(),10,5,(unsigned char *)value,      text);
        printf("Content-type: image/gif\n\n");
        gdImageGif(im_out,stdout);
        gdImageDestroy(im_out);
        return 0;
}
```

The program produces this output when called with ***cgi-bin/gdgraph1?TEXT=DavidHaskins***



While this is not the most fancy or attractive exercise but does at least demonstrate the key principles involved in generating graphical output. With the GD library you can load existing images and generate them in many formats such as **GIF**, **JPEG**, **PNG**, **WMBP** and even create **animated GIFS**.

If you have used PHP development tools you will recognise the GD library functions as they are pretty well identical showing that PHP is a wrapper in this case around the same C library.

The text displayed here is using one of the internal fonts, Tiny, Small, Medium Bold, Large and Giant which are adequate for simple labelling but you can also use TrueType fonts for more attractive output.

Geometric drawing with lines or certain styles, filled and open polygons, and circles, arcs can be created. The main thing to remember is that the origin of an image is the **TOP left hand corner** which might seem unintuitive to anyone who has studied mathematics – quite why this is I have never discovered but can only imagine that the first programmer to do anything in this area happened to not know about graphs in which we think of the origin x=0, y=0 as being at the bottom left hand.

Colours are specified as RGB values in the range 0 to 255 so that white is 255,255,255 and red is 255,0,0. To work out fine-grained hues use a graphics tool like GNU GIMP which has colour pickers so you can find out subtle RGB values. Palettes of colours from one image can be used in another and the closest colour in a palette requested or created if there is space for it to be allocated.

6.2 Using TrueType Fonts

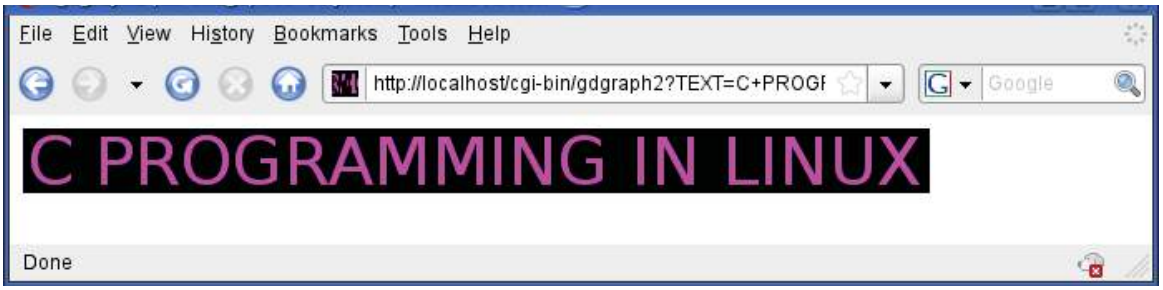
In this example we modify the previous program to generate a label using TrueType font.

```
/*****************************************************************
* C Programming in Linux (c) David Haskins 2008 * chapter6_2.c                    *
*****************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <gd.h>
#include <gdfonts.h>
#include "c_in_linux.h"

int main(int argc, char *argv[], char *env[]) {
        int text=0,background=0,height=50,width=0,x=0,y=0,size=30,string_rectangle[8];
        double angle=0.0;
        char value[255] = "";
        //OpenSuse TrueType Font
        char font[256] = "/usr/share/fonts/truetype/DejaVuSans.ttf";
        //Ubuntu TrueType Font
        // char font[256] =
        "/usr/share/fonts/truetype/ttf-dejavu/DejaVuSans.ttf";
        char *err = NULL;
        gdImagePtr im_out = NULL;

        decode_value("TEXT=", (char *) &value, 255);
        //call gdImageStringFT with NULL image to obtain size
        err = gdImageStringFT(NULL,&string_rectangle[0],0,
                font,size,angle,0,0,value);
        x = string_rectangle[2]-string_rectangle[6] + 6;
        y = string_rectangle[3]-string_rectangle[7] + 6;
        // create an image big enough for the string plus a little space
        im_out = gdImageCreate(x,y);
        //allocate colours
        background = gdImageColorAllocate(im_out, 0,0,0);
        text  = gdImageColorAllocate(im_out,255,0,255);
        //get starting position
        x = 3 - string_rectangle[6];
        y = 3 - string_rectangle[7];
        //draw the string
        err = gdImageStringFT(im_out,&string_rectangle[0],
                text,font,size,angle,x,y,value);
        //output
        printf("Content-type: image/gif\n\n");
        gdImageGif(im_out,stdout);
        gdImageDestroy(im_out);
        return 0;   }
```

We call this program as before with ***cgi-bin/gdgraph1?TEXT=C+PROGRAMMING+IN+LINUX***

to get this kind of output which you will probably see is more likely to be a useful kind of tool. The location and contents of your systems fonts will vary but the code gives an example:

- OpenSuse /usr/share/fonts/truetype/*.ttf
- Ubuntu. /usr/share/fonts/truetype/ttf-dejavu/*.ttf



To get any good at using a library like GD you have to be prepared to experiment and take a lot of time to understand the function parameters, looking in great detail at the available documentation at:

http://www.boutell.com/gd

6.3 GD function reference

A full detailed set of documentation is maintained at: http://www.boutell.com/gd

GD contains a wealth of functionality for all kinds of drawing and many formats, as well as TrueType fonts and animated Gif images. A categorised list of functions follows:

Image creation, destruction, loading and saving:

gdImageCreate(int sx, int sy)
gdImageCreateFromJpeg(FILE *in)
gdImageCreateFromPng(FILE *in)
gdImageCreateFromGif(FILE *in)
gdImageCreateFromGd(FILE *in)
gdImageCreateFromWBMP(FILE *in)
gdImageDestroy(gdImagePt rim)
void gdImageJpeg(gdImagePt rim, FILE*out, int quality)
void gdImageGif(gdImagePt rim, FILE*out)
void gdImagePng(gdImagePtr im, FILE*out)
void gdImageWBMP(gdImagePtr im, int fg, FILE*out)
void gdImageGd(gdImagePtr im, FILE*out)

Drawing Functions:

void gdImageSetPixel(gdImagePtr im, int x, int y, int color)
void gdImageLine(gdImagePtr im, int x1, int y1, int x2, int y2, int color)
void gdImageDashedLine(gdImagePtr im, int x1, int y1, int x2, int y2, int color)
void gdImagePolygon(gdImagePtr im, gdPoint Ptr point s, int point sTotal, int color)
void gdImageOpenPolygon(gdImagePtr im, gdPoint Ptr point s, int point sTotal, int color)
void gdImageRectangle(gdImagePtr im, int x1, int y1, int x2, int y2, int color)
void gdImageFilledPolygon(gdImagePtr im, gdPoint Ptr point s, int point sTotal, int color)
void gdImageFilledRectangle(gdImagePtr im, int x1, int y1, int x2, int y2, int color)
void gdImageArc(gdImagePtr im, int cx, int cy, int w, int h, int s, int e, int color)
void gdImageFilledArc(gdImagePtr im, int cx, int cy, int w, int h, int s, int e, int color, int style)
void gdImageFilledEllipse(gdImagePtr im, int cx, int cy, int w, int h, int color)
void gdImageFillToBorder(gdImagePtr im, int x, int y, int border, int color)
void gdImageFill(gdImagePtr im, int x, int y, int color)
void gdImageSetAntiAliased(gdImagePtr im, int c)
void gdImageSetAntiAliasedDontBlend(gdImagePtr im, int c)
void gdImageSetBrush(gdImagePtr im, gdImagePtr brush)

void gdImageSetTile(gdImagePtr im, gdImagePtr tile)
void gdImageSetStyle(gdImagePtr im, int *style, int styleLength)
void gdImageSetThickness(gdImagePtr im, int thickness)
void gdImageAlphaBlending(gdImagePtr im, int blending)
void gdImageSaveAlpha(gdImagePtr im, int saveFlag)
void gdImageSetClip(gdImagePtr im, int x1, int y1, int x2, int y2)
void gdImageGetClip(gdImagePtr im, int *x1P, int *y1P, int *x2P, int *y2P)

Query Functions:

int gdImageAlpha(gdImagePtr im, int color)(MACRO)
int gdImageGetPixel(gdImagePtr im, int x, int y)
int gdImageBoundsSafe(gdImagePtr im, int x, int y)
int gdImageGreen(gdImagePtr im, int color)(MACRO)
int gdImageRed(gdImagePtr im, int color)(MACRO)
int gdImageSX(gdImagePtr im)(MACRO)
int gdImageSY(gdImagePtr im)(MACRO)
int gdImageTrueColor(im)(MACRO)

Text-handling functions:

gdFontPtr gdFontGetSmall(void)
gdFontPtr gdFontGetLarge(void)
gdFontPtr gdFontGetMediumBold(void)
gdFontPtr gdFontGetGiant(void)
gdFontPtr gdFontGetTiny(void)
void gdImageChar(gdImagePtr im, gdFontPtr font, int x, int y, int c, int color)
void gdImageCharUp(gdImagePtr im, gdFontPtr font, int x, int y, int c, int color)
void gdImageString(gdImagePtr im, gdFontPtr font, int x, int y, unsigned char*s, int color)
void gdImageString16(gdImagePtr im, gdFontPtr font, int x, int y, unsigned short *s, int color)
void gdImageStringUp(gdImagePtr im, gdFontPtr font, int x, int y, unsigned char*s, int color)
void gdImageStringUp16(gdImagePtr im, gdFontPtr font, int x, int y, unsigned short*s, int color)

char *gdImageStringFT(gdImagePtr im, int *brect, int fg, char *fontname, double
ptsize, double angle, int x, int y, char*string)
char *gdImageStringFTEx(gdImagePtr im, int *brect, int fg, char *fontname, double
ptsize, double angle, int x, int y, gdFTString ExtraPtr strex)
char *gdImageStringFTCircle(gdImagePtr im, int cx, int cy, double radius,
double textRadius, double fillPortion, char*font, double point s,
char*top, char*bottom, int fgcolor)
char *gdImageStringTTF(gdImagePtr im, int *brect, int fg, char *fontname,
double ptsize, double angle, int x, int y, char *string)

Color-handling functions:

int gdImageColorAllocate(gdImagePtr im, int r, int g, int b)
int gdImageColorAllocateAlpha(gdImagePtr im, int r, int g, int b, int a)
int gdImageColorClosest(gdImagePtr im, int r, int g, int b)
int gdImageColorClosestAlpha(gdImagePtr im, int r, int g, int b, int a)
int gdImageColorClosestHWB(gdImagePtr im, int r, int g, int b)
int gdImageColorExact(gdImagePtr im, int r, int g, int b)
int gdImageColorResolve(gdImagePtr im, int r, int g, int b)
int gdImageColorResolveAlpha(gdImagePtr im, int r, int g, int b, int a)
int gdImageColorsTotal(gdImagePtr im)(MACRO)
int gdImageRed(gdImagePtr im, int c)(MACRO)
int gdImageGreen(gdImagePtr im, int c)(MACRO)
int gdImageBlue(gdImagePtr im, int c)(MACRO)
int gdImageGetInterlaced(gdImagePtr im)(MACRO)
int gdImageGetTransparent(gdImagePtr im)(MACRO)
void gdImageColorDeallocate(gdImagePtr im, int color)
void gdImageColorTransparent(gdImagePtr im, int color)
void gdTrueColor(int red, int green, int blue)(MACRO)
void gdTrueColorAlpha(int red, int green, int blue, int alpha)(MACRO)

Resizing functions:

void gdImageCopy(gdImagePtr dst, gdImagePtr src, int dstX, int dstY, int srcX, int srcY, int w, int h)
void gdImageCopyResized(gdImagePtr dst, gdImagePtr src, int dstX, int dstY, int srcX, int srcY, int destW, int destH, int srcW, int srcH)
void gdImageCopyResampled(gdImagePtr dst, gdImagePtr src, int dstX, int dstY, int srcX, Int srcY, int destW, int destH, int srcW, int srcH)
void gdImageCopyRotated(gdImagePtr dst, gdImagePtr src, doubledstX, doubledstY, int srcX, int srcY, int srcW, int srcH, int angle)
void gdImageCopyMerge(gdImagePtr dst, gdImagePtr src, int dstX, int dstY, int srcX, int srcY, int w, int h, int pct)

void gdImageCopyMergeGray(gdImagePtr dst, gdImagePtr src, int dstX, int dstY, int srcX, int srcY, int w, int h, int pct)
void gdImagePaletteCopy(gdImagePtr dst, gdImagePtr src)
void gdImageSquareToCircle(gdImagePtr im, int radius)
void gdImageSharpen(gdImagePtr im, int pct)

Miscellaneous Functions:

int gdImageCompare(gdImagePtr im1, gdImagePtr im2)
gdImageInterlace(gdImagePtr im, int int erlace)
gdFree(void *ptr)

In order to use a library like this you will need familiarity with the arguments which are often data types defined within the library itself such as gdmagePtr which is a pointer to some kind of structure containing all the data for an image to be processed or stored. These may all seem unusual but after a while you will begin to get used to the syntax and on-line documentation and begin to see patterns in the complexity.