# Image Processing using GTK.

Muthiah Annamalai

June 2, 2004

## Contents

[1]This Document and Code are realeased under GNU FDL & GPL repectively

# 1 Introduction

Keywords: GTK,GDK,Image Processing,Filters,Convolution,Pixel
Latest versions of the document can be had from http://cybernetics.freewebspace.com/gtk/

*For the eaxmple code visit http://cybernetics.freewebspace.com/gtk/*
in case you cant some code is attached below, in the appendix.


Figure 1: GTK Tool Kit.

**Abstract: This document aims to explain the feasibilty of image processing using GTK, the GIMP Tool Kit, which is a free software graphics toolkit useful for GUI programming. GTK is also freely available for the GNU/Linux and many other platforms.**
**This document however just scratches the surface when it comes to explaining the nook & corner of Image processing theory. It shows eaxmples of pixel based filters and convolution filters.**

## 1.1 GTK & GDK

The Gimp Tool Kit [GTK], is a popular GUI programm library omnipresent in the world of free software.It helps us make windows, user interfaces like dialogs, buttons, menus and a plethora of features. Here however, we will focus on the ease of loading, manipulating and displaying images or pictures present in various formats using the GTK toolkit's drawing libraries called Graphics Drawing Kit [GDK]. Though it would be very tempting to introduce GTK right now, its left to the reader to pursue his/her curiosity to learn GTK.

The outline of what we wil do is this; we use the GDK library to load images and then get the pixel buffer of the image, and manipulate it. Afterwards we make this image into a GTK widget and display it on the screen. Just three simple steps;

- load,
- process,
- display.

Presto! Welcome to Image Processing.

## 1.2 Image Processing

The idea of image processing. Generally it may so happen that we desire image,photos and pictures of real world at a higher clarity, better quality and optimum colour levels that we are not satisfied by the original images we get from our cameras, electronic as well as analog, so we find a pressing need to iron out the creases, remove the crow's feet from our photos and convert black and white pictures into colored ones. Welcome to the world of Image processing. These are the precise tools that will help solve many of the above problems.

## 1.3 Digital ImageProcessing

The digital [or computer] representation of images happens like this; A digital camera captures the image, by converting the light intensity to equivalent electrical voltage levels and this is quantized by and ADC [Analog to Digial Converter], and the resulting data is stored in the camera's digital memory. The essential process that happens is that of quantizing, the real colour to a small digital value, and at that ensuring we keep it small enough to be stored in memory and large enough to hold all the colours we desire. Thus evolved the generic term for a colour element on the computer: Pixel, [Picture Element], which stores the values of colour components, namely red,green and blue. For the programmer a pixel can be represented like this:

```
struct pixel
{
unsigned char red;    // 0  to   255
unsigned char green;  // 0  to   255
unsigned char blue;   // 0  to   255
};
```

Here the pixel is large enough to allow us represent some 16777216 [16 Million Colours] approximately, and small enough to represent it in 24 bits! An image is a thus, intuitively a collection of different colours arragned along different rows and columns. The colours are nothing but pixels; which follows that an image is a collection of pixels in rows and columns. A image to us is simply an 2-D array of pixels.

Now that you have understood what an image is, and how its represented in an array of pixels, various formats exist in the wide world of image processing to encode and store the images, so that they occupy the least amount of memory, to facilitate among other things faster downloads and quicker processing. These formats, which you may be familiar are the ones like *jpeg, png, bmp, xpm* etc. Now each format uses various techniques for encoding; for example jpeg use *DCT [Discrete Cosine Transform]*, bmp can be *RLE [Run Length Encoded*], and PNG could use dictionary based compression. Whew so much for a 2-D array of pixels, that the weak hearted might consider quitting.Wait; thats why the GTK and GDK libraries are considered the panacea to this pain. GDK has function calls that loads from almost any image to the sought after 2-D array of pixels.And need I say more? Lets start.

GDK can load files using the function *gdk_pixbuf_new_from_file* and we can get various attributes of the image like width,height and pointer to pixel buffer(to manipulate it!),process the image next, and then we can make a GTK widget from it. The following section of commented code walks through a typical section.

1. **Attribute function**
2. width *int gdk_pixxbuf_get_width*
3. height *int gdk_pixxbuf_get_height*
4. pixel pointer *gchar *gdk_pixxbuf_get_pixels*
5. rowstride *int gdk_pixxbuf_get_rowstride*

```
  /*
Code for loading and image & displaying it.
  */
  #include<gtk/gtk.h>

  gchar *filename="/usr/share/pixmaps/gnu_emacs.png"; //picture to load
  GdkPixbuf *pb;                                     //pixbuf structure
  GtkImage *im;                                      //gtk widget

  pb=gdk_pixbuf_new_from_file(filename,NULL);        //load the image to pixbuf

  process_picture(pb);      //process it the fun begins here!

  im=gtk_image_new_from_pixbuf(pb);      //make it a widget

  gtk_widget_show(im);      //display the image widget
```

I will assume that from this point onwards, we are working with a standard pixel, as defined above in section [see pixel] Of the many effets possible after befriending the pixel, the easiest ones are the pixel level or point filters.They consist of, but not limited to

1. Grayscaling see sec 3
2. ColourInverting see sec 1.3
3. Colour Masking see sec 2
4. Brighten see sec 5
5. Flip see sec 6
6. Invert see sec 7
7. Border see sec 8

A note of accessing the pixel array as returned by the GDK. As of writing the GDK supports image of the type RGBA or RGB only. Hence we are supposed to get the attributes of height,width and bytes per pixel, before we begin. The pointer to the pixel array is a 1-D array of pixels, which has to navigated like a 2-D array as show below. It has at offsets of 0,1,2 form the current pixel location, the values of the red, green, and blue values of the current pixel.

```
int ht,wt;
int i=0,j=0;
int rowstride=0;
int bpp=0;
gchar *pixel;


if(gdk_pixbuf_get_bits_per_sample(pb)!=8)   //we handle only 24 bit images.
 return;                                    //look at 3 bytes per pixel.

//getting attributes of height,
ht=gdk_pixbuf_get_height(pb);   //width, and bpp.Also get pointer
wt=gdk_pixbuf_get_width(pb);   //to pixels.
pixel=gdk_pixbuf_get_pixels(pb);

bpp=3;
rowstride=wt*bpp;

for(i=0;i<ht;i++) //iterate over the height of image.
  for(j=0;j<rowstride;j+=bpp)   //read every pixel in the row.skip
```

```
//bpp bytes
      {

      //access pixel[i][j] as
      // pixel[i*rowstride + j]

      //access red,green and blue as

      pixel[i*rowstride + j+0]=red
      pixel[i*rowstride + j+1]=green
      pixel[i*rowstride + j+2]=blue

      }
```

We will be using the figure of the kathakali dancer shown here in figure, for our iamge processing needs.


Figure 2: A Kathakali dancer. Kathakali is the traditional art form of Kerala [India]

## 2 Colour Inverting:

This is the simple operation of making every pixel take its 1's complement. This in effect creates the negative of the image given. We can do this by loading each pixel and doing the following transformation.

```
    pixel.red   =255-pixel.red;
pixel.green =255-pixel.green;
pixel.blue  =255-pixel.blue;
```

Accessing the image pixels as shown in the above code section 2 we use the transformation inside the loop. This code below in sec 2 yielded this image.


Figure 3: Kathakali, colour inverted.

```
#include<gtk/gtk.h>

void colorinvert_picture(GdkPixbuf *pb)
{
  int ht,wt;
  int i=0,j=0;
  int rowstride=0;
  int bpp=0;
```

```
  gchar *pixel;


  if(gdk_pixbuf_get_bits_per_sample(pb)!=8)   //we handle only 24 bit images.
   return;                                    //look at 3 bytes per pixel.

  bpp=3;              //getting attributes of height,
  ht=gdk_pixbuf_get_height(pb);   //width, and bpp.Also get pointer
  wt=gdk_pixbuf_get_width(pb);   //to pixels.
  pixel=gdk_pixbuf_get_pixels(pb);
  rowstride=wt*bpp;

  for(i=0;i<ht;i++) //iterate over the height of image.
    for(j=0;j<rowstride;j+=bpp)   //read every pixel in the row.skip
//bpp bytes
      {

      //access pixel[i][j] as
      // pixel[i*rowstride + j]

      //access red,green and blue as
pixel[i*rowstride + j+0]=255-pixel[i*rowstride + j+0];
pixel[i*rowstride + j+1]=255-pixel[i*rowstride + j+1];
pixel[i*rowstride + j+2]=255-pixel[i*rowstride + j+2];
      }
  return;
}

% <!-- provide link here-->
% complete working example with glue code in example_colorinvert.c
```


Figure 4: Kathakali, Red masked.

## 3 Colour Masking.

Following similar precept we can do color masking by removing the effect of that particular colour. I will show you the masking for red as the masking for other colours is obvious.

```
    pixel.red   =0x00; //mask red.
pixel.green |=0; //no change.
pixel.blue  |=0; //no change.
```

The routine below modifies the image removing the said component [color takes values of either [r/R] or [g/G] or [b/B]] and processes it.[ see 3] . This code yielded figures like this.

Figure 5: Kathakali, Green masked.

Figure 6: Kathakali, Blue masked.

```
#include<gtk/gtk.h>

void colormask_picture(GdkPixbuf *pb,gchar color)
{

  int ht,wt;
  int i=0,j=0;
  int rowstride=0;
  int bpp=0;
  gchar *pixel;


  if(gdk_pixbuf_get_bits_per_sample(pb)!=8)   //we handle only 24 bit images.
   return;                                    //look at 3 bytes per pixel.

  bpp=3;              //getting attributes of height,
  ht=gdk_pixbuf_get_height(pb);   //width, and bpp.Also get pointer
  wt=gdk_pixbuf_get_width(pb);   //to pixels.
  pixel=gdk_pixbuf_get_pixels(pb);
  rowstride=wt*bpp;
```

```
  for(i=0;i<ht;i++) //iterate over the height of image.
    for(j=0;j<rowstride;j+=bpp)   //read every pixel in the row.skip
//bpp bytes
      {

      //access pixel[i][j] as
      // pixel[i*rowstride + j]

      //access red,green and blue as
switch(color)
  {
  case 'r':
  case 'R':
    pixel[i*rowstride + j+0]=0x00;//mask red;
    break;
  case 'g':
  case 'G':
    pixel[i*rowstride + j+1]=0x00;//mask green
    break;
  case 'b':
  case 'B':
    pixel[i*rowstride + j+2]=0x00;//mask blue;
  default:
    pixel[i*rowstride + j+2]=0x00;//mask blue;
  }
      }
  return;
}

% <!-- provide link here-->
% complete working example with glue code in example_colormask.c
```

# 4 Grey Scaling.

Making all the three pixels take up the value of their average values gives a gray scale effect. This however is very rudimentary, method of grey scaling. *The image looks very ugly so Im not putting it up. ;-)!*

```
average=(pixel.red+pixel.green+pixel.blue)/3;
        pixel.red   =average;
        pixel.green =average;
        pixel.blue  =average;
```

```
#include<gtk/gtk.h>

void colorgrey_picture(GdkPixbuf *pb)
{
  int ht,wt;
  int i=0,j=0;
  int rowstride=0;
  int bpp=0;
  double avg=0;
  gchar *pixel;


  if(gdk_pixbuf_get_bits_per_sample(pb)!=8)   //we handle only 24 bit images.
   return;                                    //look at 3 bytes per pixel.

  bpp=3;              //getting attributes of height,
  ht=gdk_pixbuf_get_height(pb);   //width, and bpp.Also get pointer
  wt=gdk_pixbuf_get_width(pb);   //to pixels.
  pixel=gdk_pixbuf_get_pixels(pb);
  rowstride=wt*bpp;

  for(i=0;i<ht;i++) //iterate over the height of image.
    for(j=0;j<rowstride;j+=bpp)   //read every pixel in the row.skip
//bpp bytes
      {

//find avg of the pixel to grey it.

avg+=pixel[i*rowstride + j+0]+pixel[i*rowstride + j+1]+pixel[i*rowstride + j+2];
avg/=3.00;
avg=((int)(avg))%256;
```

```
pixel[i*rowstride + j+0]=(int)avg;
pixel[i*rowstride + j+1]=(int)avg;
pixel[i*rowstride + j+2]=(int)avg;
      }
  return;
}

% <!-- provide link here-->
% complete working example with glue code in example_colorgrey.c
```

Figure 7: Kathakali, on the MSB bit 7

# 5 Bitplaning:

Every pixel is make up of 3 bytes which contribute various values to the picture, which can be ascertained from the weigth of their pixels. As we can guess the MSB has the greatest weight and the others carry lesser weigth, in the binary sequence.

By finding the image formed by the LSB bit to the MSB bit we can appreciate the levels of importance of a picture in its bytes.

```
mask=(0xff & (1<<plane));

        pixel.red   &=mask;
        pixel.green &=mask;
        pixel.blue  &=mask;
```

Calling the function colorbitplanize_picture(pb,7); with the bitplane yields the results as shown. The plane 7 [MSB] is the highest while plane 0 is the lowest [LSB].

```
#include<gtk/gtk.h>
```

```
void colorbitplanize_picture(GdkPixbuf *pb,guchar plane)
{
  int ht,wt;
  int i=0,j=0;
  int rowstride=0;
  int bpp=0;
  guchar mask=0;
  gchar *pixel;


  if(gdk_pixbuf_get_bits_per_sample(pb)!=8)   //we handle only 24 bit images.
   return;                                    //look at 3 bytes per pixel.

  bpp=3;              //getting attributes of height,
  ht=gdk_pixbuf_get_height(pb);   //width, and bpp.Also get pointer
  wt=gdk_pixbuf_get_width(pb);   //to pixels.
  pixel=gdk_pixbuf_get_pixels(pb);
  rowstride=wt*bpp;

  mask=(0xff&(1<<plane));

  for(i=0;i<ht;i++) //iterate over the height of image.
    for(j=0;j<rowstride;j+=bpp)   //read every pixel in the row.skip
//bpp bytes
      {
//find avg of the pixel to grey it.

pixel[i*rowstride + j+0]&=mask;
pixel[i*rowstride + j+1]&=mask;
pixel[i*rowstride + j+2]&=mask;
      }
  return;
}

% <!-- provide link here-->
% complete working example with glue code in example_colorplane.c
```

# 6 Brightening:

Just add an offset to the original image's pixel components[Red,Green,Blue] and then make it look like really bright. Please tweak with the value of offset to get a decent level of brightness.

```
        pixel.red   += offset;
        pixel.green += offset;
        pixel.blue  += offset;
```

*some one please fix it! This effect is really bad!*

```
#include<gtk/gtk.h>


void colorbright_picture(GdkPixbuf *pb,int offset)
{
  int ht,wt;
  int i=0,j=0;
  int rowstride=0;
  int bpp=0;
  gchar *pixel;


  if(gdk_pixbuf_get_bits_per_sample(pb)!=8)   //we handle only 24 bit images.
   return;                                    //look at 3 bytes per pixel.

  bpp=3;              //getting attributes of height,
  ht=gdk_pixbuf_get_height(pb);   //width, and bpp.Also get pointer
  wt=gdk_pixbuf_get_width(pb);   //to pixels.
  pixel=gdk_pixbuf_get_pixels(pb);
  rowstride=wt*bpp;

  for(i=0;i<ht;i++) //iterate over the height of image.
    for(j=0;j<rowstride;j+=bpp)   //read every pixel in the row.skip
//bpp bytes
      {

//Manipulate Red.
if((pixel[i*rowstride + j+0]+offset) >= 0xff)
  {
    pixel[i*rowstride + j+0]=0xff;
  }
```

```
else if((pixel[i*rowstride + j+0]-offset) < 0)
  {
    pixel[i*rowstride + j+0]=0;
  }
else
  pixel[i*rowstride + j+0]+=offset;


//Manip Green
if((pixel[i*rowstride + j+1]+offset) >= 0xff)
  {
    pixel[i*rowstride + j+1]=0xff;
  }
else if((pixel[i*rowstride + j+1]-offset) < 0)
  {
    pixel[i*rowstride + j+1]=0;
  }
else
  pixel[i*rowstride + j+1]+=offset;


//Manip Blue
if((pixel[i*rowstride + j+2]+offset) >= 0xff)
  {
    pixel[i*rowstride + j+2]=0xff;
  }
else if((pixel[i*rowstride + j+2]-offset) < 0)
  {
    pixel[i*rowstride + j+2]=0;
  }
else
  pixel[i*rowstride + j+2]+=offset;
      }
  return;
}
% <!-- provide link here-->
% complete working example with glue code in example_bright.c
```

Figure 8: Kathakali, Image L-R flipped, Look closely! see Fig:2

# 7 Image flipping

Take the first column and swap it with the last col. Do this for the first width/2 columns then we have a flipped image.

```
void flip_picture(GdkPixbuf *pb) //Left to Right
{
  int ht,wt;
  int i=0,j=0;

  gchar *pixel;
  ht=gdk_pixbuf_get_height(pb);
  wt=gdk_pixbuf_get_width(pb);
  pixel=gdk_pixbuf_get_pixels(pb);



  for(i=0;i<ht;i++)           //half the width
    for(j=0;j<(wt/2)*3;j+=3)
      {
swap(pixel[i*wt*3+j+0],pixel[i*wt*3+(wt*3-j)+0]);
```

```
swap(pixel[i*wt*3+j+1],pixel[i*wt*3+(wt*3-j)+1]);
swap(pixel[i*wt*3+j+2],pixel[i*wt*3+(wt*3-j)+2]);
      }
  return;
}
```

Figure 9: Kathakali, Image inverted

## 8 Image Inversion

Take the first row a swap it with the last row. Do this for the first height/2 rows then we have a inverted swapped image.

```
#define swap(x,y) {int temp;temp=x;x=y;y=temp;}

void invert_picture(GdkPixbuf *pb) //ulta 180* degree turn
{
  int ht,wt;
  int i=0,j=0;
  int rowstride=0;

  gchar *pixel;
  ht=gdk_pixbuf_get_height(pb);
  wt=gdk_pixbuf_get_width(pb);
  pixel=gdk_pixbuf_get_pixels(pb);
  rowstride=gdk_pixbuf_get_rowstride(pb);


  for(i=0;i<ht/2;i++)           //half the height!
```

```
    for(j=0;j<rowstride;j+=3)
      {
swap(pixel[i*rowstride+j+0],pixel[(ht-i)*rowstride+j+0]);
swap(pixel[i*rowstride+j+1],pixel[(ht-i)*rowstride+j+1]);
swap(pixel[i*rowstride+j+2],pixel[(ht-i)*rowstride+j+2]);
      }
  return;
}
```


Figure 10: Kathakali, With Border

# 9 Drawing Border

Draws a simple border of the given colour around the image. It destroys the pixels present there though!

```
void draw_normal_border(GdkPixbuf *pb,int red,int green,int blue)
{
  int ht,wt;
  int i=0,j=0;
  int flag=0;
  int border=10; //10 pixels wide border.
  int rowstride=0;
  gchar *pixel;

  ht=gdk_pixbuf_get_height(pb);
  wt=gdk_pixbuf_get_width(pb);
  pixel=gdk_pixbuf_get_pixels(pb);
  rowstride=gdk_pixbuf_get_rowstride(pb);


  for(i=0;i<ht;i++)
```

```
    for(j=0;j<rowstride;j+=3)
      {
flag=0;//reset flag every time.

if(i< (border))
  flag=1;
else if(i > (ht-border))
  flag=1;
else if(j < (border)*3)
  flag=1;
else if(j > (wt-border)*3)
  flag=1;


if(flag==1)
  {
    pixel[i*rowstride+j+0]=red;
    pixel[i*rowstride+j+1]=green;
    pixel[i*rowstride+j+2]=blue;
  }
      }
  return;
}
```

See you soon!
Cheers
Muthu
The intricacies of GTK are too good!

Appendix

# A Border: example_colorborder.c

```
#include<gtk/gtk.h>
#include<glib/gprintf.h>
#include<string.h>


void draw_normal_border(GdkPixbuf *pb,int red,int green,int blue)
{
  int ht,wt;
  int i=0,j=0;
  int flag=0;
  int border=10; //10 pixels wide border.
  int rowstride=0;
  gchar *pixel;

  ht=gdk_pixbuf_get_height(pb);
  wt=gdk_pixbuf_get_width(pb);
  pixel=gdk_pixbuf_get_pixels(pb);
  rowstride=gdk_pixbuf_get_rowstride(pb);


  for(i=0;i<ht;i++)
    for(j=0;j<rowstride;j+=3)
      {
flag=0;//reset flag every time.

if(i< (border))
  flag=1;
else if(i > (ht-border))
  flag=1;
else if(j < (border)*3)
```

```
  flag=1;
else if(j > (wt-border)*3)
  flag=1;


if(flag==1)
  {
    pixel[i*rowstride+j+0]=red;
    pixel[i*rowstride+j+1]=green;
    pixel[i*rowstride+j+2]=blue;
  }
      }
  return;
}

GtkWidget* get_image(const gchar *filename)
{

  GtkWidget *im;
  GdkPixbuf *pb;
  gchar *savefile,*file;
  pb=gdk_pixbuf_new_from_file(filename,NULL);

  draw_normal_border(pb,0xff,00,0xff);

  //ignore the extension and save as jpeg.
  file=g_strdup(filename);
  savefile=strchr(file,'.');
  if(savefile!=NULL)
    *savefile='\0';
  savefile=g_strdup_printf("%s%s",file,"_colorborder.jpeg");

  gdk_pixbuf_save(pb,savefile,"jpeg",NULL,NULL);//save the file
  im=gtk_image_new_from_pixbuf(pb);
```

```
  gdk_pixbuf_unref(pb);

  g_free(savefile);
  g_free(file);
  return im;
}


int main(int argc,char *argv[])
{
  GtkWidget *w,*vb,*b;
  gchar *file="kathakali.jpeg";
  gtk_init(&argc,&argv);
  w=gtk_window_new(GTK_WINDOW_TOPLEVEL);
  gtk_window_set_default_size(GTK_WINDOW(w),200,200);

  b=gtk_button_new_with_mnemonic("_Exit");

  vb=gtk_vbox_new(FALSE,FALSE);
  gtk_container_add(GTK_CONTAINER(w),vb);
  gtk_box_pack_start_defaults(GTK_BOX(vb),get_image(file));
  gtk_box_pack_start_defaults(GTK_BOX(vb),gtk_hseparator_new());

  gtk_box_pack_start_defaults(GTK_BOX(vb),gtk_image_new_from_file(file));
  gtk_box_pack_start_defaults(GTK_BOX(vb),b);//exit button.


  gtk_signal_connect(GTK_OBJECT(w),"destroy",GTK_SIGNAL_FUNC(gtk_main_quit),NU
  gtk_signal_connect(GTK_OBJECT(b),"clicked",GTK_SIGNAL_FUNC(gtk_main_quit),NU

  gtk_widget_show_all(w);

  gtk_main();
  return 0;
```

```
}

//compilation lines
//gcc -Wall example_colorborder.c `pkg-config libgnomeui-2.0 --cflags --libs`
```

## B Colour Invert: example_colorinvert.c

```
#include<gtk/gtk.h>
#include<glib/gprintf.h>
#include<string.h>

void colorinvert_picture(GdkPixbuf *pb)
{
  int ht,wt;
  int i=0,j=0;
  int rowstride=0;
  int bpp=0;
  gchar *pixel;


  if(gdk_pixbuf_get_bits_per_sample(pb)!=8)   //we handle only 24 bit images.
   return;                                    //look at 3 bytes per pixel.

  bpp=3;              //getting attributes of height,
  ht=gdk_pixbuf_get_height(pb);   //width, and bpp.Also get pointer
  wt=gdk_pixbuf_get_width(pb);   //to pixels.
  pixel=gdk_pixbuf_get_pixels(pb);
  rowstride=wt*bpp;

  for(i=0;i<ht;i++) //iterate over the height of image.
    for(j=0;j<rowstride;j+=bpp)   //read every pixel in the row.skip
//bpp bytes
      {

      //access pixel[i][j] as
      // pixel[i*rowstride + j]

      //access red,green and blue as
pixel[i*rowstride + j+0]=255-pixel[i*rowstride + j+0];
```

```
pixel[i*rowstride + j+1]=255-pixel[i*rowstride + j+1];
pixel[i*rowstride + j+2]=255-pixel[i*rowstride + j+2];
      }
  return;
}

GtkWidget* get_image(const gchar *filename)
{

  GtkWidget *im;
  GdkPixbuf *pb;
  gchar *savefile,*file;
  pb=gdk_pixbuf_new_from_file(filename,NULL);

  colorinvert_picture(pb);

  //ignore the extension and save as jpeg.
  file=g_strdup(filename);
  savefile=strchr(file,'.');
  if(savefile!=NULL)
    *savefile='\0';
  savefile=g_strdup_printf("%s%s",file,"_colorinvert.jpeg");

  gdk_pixbuf_save(pb,savefile,"jpeg",NULL,NULL);//save the file
  im=gtk_image_new_from_pixbuf(pb);
  gdk_pixbuf_unref(pb);

  g_free(savefile);
  g_free(file);
  return im;
}


int main(int argc,char *argv[])
```

```
{
  GtkWidget *w,*vb,*b;
  gchar *file="kathakali.jpeg";
  gtk_init(&argc,&argv);
  w=gtk_window_new(GTK_WINDOW_TOPLEVEL);
  gtk_window_set_default_size(GTK_WINDOW(w),200,200);

  b=gtk_button_new_with_mnemonic("_Exit");

  vb=gtk_vbox_new(FALSE,FALSE);
  gtk_container_add(GTK_CONTAINER(w),vb);
  gtk_box_pack_start_defaults(GTK_BOX(vb),get_image(file));
  gtk_box_pack_start_defaults(GTK_BOX(vb),gtk_hseparator_new());

  gtk_box_pack_start_defaults(GTK_BOX(vb),gtk_image_new_from_file(file));
  gtk_box_pack_start_defaults(GTK_BOX(vb),b);//exit button.


  gtk_signal_connect(GTK_OBJECT(w),"destroy",GTK_SIGNAL_FUNC(gtk_main_quit),NU
  gtk_signal_connect(GTK_OBJECT(b),"clicked",GTK_SIGNAL_FUNC(gtk_main_quit),NU

  gtk_widget_show_all(w);

  gtk_main();
  return 0;
}

//compilation lines
//gcc -Wall example_colorinvert.c `pkg-config libgnomeui-2.0 --cflags --libs`
```

## C Image Flip: example_pictureflip.c

```
#include<gtk/gtk.h>
#include<glib/gprintf.h>
#include<string.h>

#define swap(x,y) {int temp;temp=x;x=y;y=temp;}

void flip_picture(GdkPixbuf *pb) //L-R swap
{
  int ht,wt;
  int i=0,j=0;
  int rowstride=0;
  gchar *pixel;

  ht=gdk_pixbuf_get_height(pb);
  wt=gdk_pixbuf_get_width(pb);
  pixel=gdk_pixbuf_get_pixels(pb);
  rowstride=gdk_pixbuf_get_rowstride(pb);


  for(i=0;i<ht;i++)           //half the width
    for(j=0;j<(rowstride)/2;j+=3)
      {
swap(pixel[i*rowstride+j+0],pixel[i*rowstride+(rowstride-j)+0]);
swap(pixel[i*rowstride+j+1],pixel[i*rowstride+(rowstride-j)+1]);
swap(pixel[i*rowstride+j+2],pixel[i*rowstride+(rowstride-j)+2]);
      }
  return;
}

GtkWidget* get_image(const gchar *filename)
{
```

```
  GtkWidget *im;
  GdkPixbuf *pb;
  gchar *savefile,*file;
  pb=gdk_pixbuf_new_from_file(filename,NULL);

  flip_picture(pb);

  //ignore the extension and save as jpeg.
  file=g_strdup(filename);
  savefile=strchr(file,'.');
  if(savefile!=NULL)
    *savefile='\0';
  savefile=g_strdup_printf("%s%s",file,"_pictureflip.jpeg");

  gdk_pixbuf_save(pb,savefile,"jpeg",NULL,NULL);//save the file
  im=gtk_image_new_from_pixbuf(pb);
  gdk_pixbuf_unref(pb);

  g_free(savefile);
  g_free(file);
  return im;
}


int main(int argc,char *argv[])
{
  GtkWidget *w,*vb,*b;
  gchar *file="kathakali.jpeg";
  gtk_init(&argc,&argv);
  w=gtk_window_new(GTK_WINDOW_TOPLEVEL);
  gtk_window_set_default_size(GTK_WINDOW(w),200,200);

  b=gtk_button_new_with_mnemonic("_Exit");
```

```
  vb=gtk_vbox_new(FALSE,FALSE);
  gtk_container_add(GTK_CONTAINER(w),vb);
  gtk_box_pack_start_defaults(GTK_BOX(vb),get_image(file));
  gtk_box_pack_start_defaults(GTK_BOX(vb),gtk_hseparator_new());

  gtk_box_pack_start_defaults(GTK_BOX(vb),gtk_image_new_from_file(file));
  gtk_box_pack_start_defaults(GTK_BOX(vb),b);//exit button.


  gtk_signal_connect(GTK_OBJECT(w),"destroy",GTK_SIGNAL_FUNC(gtk_main_quit),NU
  gtk_signal_connect(GTK_OBJECT(b),"clicked",GTK_SIGNAL_FUNC(gtk_main_quit),NU

  gtk_widget_show_all(w);

  gtk_main();
  return 0;
}
//compilation lines
//gcc -Wall example_imageflip.c `pkg-config libgnomeui-2.0 --cflags --libs`
```

# D License

This document and code are released under GNU FDL and GPL respectively

```
Image Processing Using GTK
Copyright (C) 2004 Muthiah Annamalai

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 2 of the License, or
(at your option) any later version.

This program 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.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
```

# E Disclaimer

Im not responsible for any crazy thing that might happen because of doing whats in here! Love GPL.

# F Author

Latest versions of the document can be had from
http://cybernetics.freewebspace.com/gtk/

Please send comments & criticisms to:
Contact me at: dearestchum@yahoo.co.in
About me: Im a guy, 20 something, love GNU, full time engineer, and working towards a degree in Electronics in India. Its Happening here!
My Picture:

Figure 11: I Love Mandrake GNU Linux!


Figure 12: GTK Book Initiative.