To install gd package:

sudo apt-get install libgd2-xpm-dev

To compile: gcc –fopenmp input.png output.png 4

Converting a color image into negative is very simple. All we have to do is repeat 3 simple steps for each pixels of the image.

1. Get the RGB value of the pixel.
2. Calculate the new RGB value as shown below.  
     
   R = 255 – R  
   G = 255 – G  
   B = 255 – B
3. Save the new RGB value in the pixel.

Example:

Consider a color pixel with the following values

R = 100

G = 150

B = 200

Where R, G and B represents the Red, Green and Blue value of the pixel.

Remember! RGB will have an integer value in the range 0 to 255.

So, to convert the color pixel into negative we will subtract the value of R, G and B from 255.

R = 255 - 100 = 155

G = 255 - 150 = 105

B = 255 - 200 = 55

So, the new RGB value will be:

R = 155

G = 105

B = 55

**Converting a color image into black and white image:**

When an image is black and white it has the same R,G,B values. So in a colored image if (R,G,B) = (10,20,30), but in black and white it should be (10+20+30)/3 = 20, so Black and white image would have values of (20,20,20)

Details of gd packages:

|  |  |
| --- | --- |
| [gdImageGetPixel](https://libgd.github.io/manuals/2.2.5/files/gd-c.html#gdImageGetPixel) | Gets a pixel color as stored in the image. |

**gdImageSX**

Gets the width (in pixels) of an image.

**Parameters**

|  |  |
| --- | --- |
| im | The image. |
|  |  |

**gdImageSY**

Gets the height (in pixels) of an image.

**Parameters**

|  |  |
| --- | --- |
| im | The image. |

[gdImageCreateFromPng](https://libgd.github.io/manuals/2.1.1/files/gd_png-c.html#gdImageCreateFromPng) is called to load images from PNG format files.  Invoke [gdImageCreateFromPng](https://libgd.github.io/manuals/2.1.1/files/gd_png-c.html" \l "gdImageCreateFromPng) with an already opened pointer to a FILE containing the desired image.  [gdImageCreateFromPng](https://libgd.github.io/manuals/2.1.1/files/gd_png-c.html" \l "gdImageCreateFromPng) returns a [gdImagePtr](https://libgd.github.io/manuals/2.1.1/files/gd-h.html" \l "gdImagePtr) to the new image, or NULL if unable to load the image (most often because the file is corrupt or does not contain a PNG image).  [gdImageCreateFromPng](https://libgd.github.io/manuals/2.1.1/files/gd_png-c.html" \l "gdImageCreateFromPng) does not close the file.  You can inspect the sx and sy members of the image to determine its size.  The image must eventually be destroyed using gdImageDestroy().

|  |  |
| --- | --- |
| [gdImageRed](https://libgd.github.io/manuals/2.2.4/files/gd-h.html#gdImageRed) | Gets the red component value of a given color. |
| [gdImageGreen](https://libgd.github.io/manuals/2.2.4/files/gd-h.html#gdImageGreen) | Gets the green component value of a given color. |
| [gdImageBlue](https://libgd.github.io/manuals/2.2.4/files/gd-h.html#gdImageBlue) | Gets the blue component value of a given color. |

**gdImageColorAllocate**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | | int gdImageColorAllocate ( | gdImagePtr | im, | |  | int | r, | |  | int | g, | |  | int | b | ) | |

Allocates a color

**Parameters**

|  |  |
| --- | --- |
| im | The image. |
| r | The value of the red component. |
| g | The value of the green component. |
| b | The value of the blue component. |

|  |  |
| --- | --- |
| [gdImagePngEx](https://libgd.github.io/manuals/2.1.1/files/gd_png-c.html#gdImagePngEx) | [gdImagePngEx](https://libgd.github.io/manuals/2.1.1/files/gd_png-c.html#gdImagePngEx) outputs the specified image to the specified file in PNG format. |
| [gdImagePng](https://libgd.github.io/manuals/2.1.1/files/gd_png-c.html#gdImagePng) | Equivalent to calling [gdImagePngEx](https://libgd.github.io/manuals/2.1.1/files/gd_png-c.html" \l "gdImagePngEx) with compression of -1. |

**gdImageSetPixel**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | | void gdImageSetPixel ( | gdImagePtr | im, | |  | int | x, | |  | int | y, | |  | int | color | ) | |

#include <stdio.h>

#include <gd.h>

#include <string.h>

#include <omp.h>

int main(int argc, char \*argv[]) {

if (argc < 4) {

printf("Usage: ./negative input.png output.png num\_threads\n");

return 1;

}

char \*input\_file = argv[1];

char \*output\_file = argv[2];

int num\_threads = atoi(argv[3]);

int color, x, y, i;

int red, green, blue;

FILE \*fp;

if((fp = fopen(input\_file, "r")) == NULL) {

printf("Error opening file %s\n", input\_file);

return 1;

}

gdImagePtr img = gdImageCreateFromPng(fp);

int width = gdImageSX(img);

int height = gdImageSY(img);

double t1 = omp\_get\_wtime();

#pragma omp parallel for private(y, color, red, green, blue) num\_threads(num\_threads)

for(x=0; x<width; x++) {

for(y=0; y<height; y++) {

color = x + 0;

color = gdImageGetPixel(img, x, y);

red = 255 - gdImageRed(img, color);

green = 255 - gdImageGreen(img, color);

blue = 255 - gdImageBlue(img, color);

color = gdImageColorAllocate(img, red, green, blue);

gdImageSetPixel(img, x, y, color);

}

}

double t2 = omp\_get\_wtime();

if((fp=fopen(output\_file, "w")) == NULL) {

printf("Error opening output file %s\n", output\_file);

return 1;

}

gdImagePng(img, fp);

gdImageDestroy(img);

fclose(fp);

printf("File Size: %dx%d\n", width, height);

printf("Time Taken: %.3lfms\n",(t2 - t1) \* 1000);

return 0;

}

**Using Critical Section**

#include <stdio.h>

#include <gd.h>

#include <string.h>

#include <omp.h>

int main(int argc, char \*argv[]) {

if (argc < 4) {

printf("Usage: ./negative input.png output.png num\_threads\n");

return 1;

}

char \*input\_file = argv[1];

char \*output\_file = argv[2];

int num\_threads = atoi(argv[3]);

int color, x, y, i;

int red, green, blue;

FILE \*fp;

if((fp = fopen(input\_file, "r")) == NULL) {

printf("Error opening file %s\n", input\_file);

return 1;

}

gdImagePtr img = gdImageCreateFromPng(fp);

int width = gdImageSX(img);

int height = gdImageSY(img);

double t1 = omp\_get\_wtime();

#pragma omp parallel for private(y, color, red, green, blue) num\_threads(num\_threads)

for(x=0; x<width; x++) {

#pragma omp critical

{

for(y=0; y<height; y++) {

color = x + 0;

color = gdImageGetPixel(img, x, y);

red = 255 - gdImageRed(img, color);

green = 255 - gdImageGreen(img, color);

blue = 255 - gdImageBlue(img, color);

color = gdImageColorAllocate(img, red, green, blue);

gdImageSetPixel(img, x, y, color);

}

}

}

double t2 = omp\_get\_wtime();

if((fp=fopen(output\_file, "w")) == NULL) {

printf("Error opening output file %s\n", output\_file);

return 1;

}

gdImagePng(img, fp);

gdImageDestroy(img);

fclose(fp);

printf("File Size: %dx%d\n", width, height);

printf("Time Taken: %.3lf ms\n",(t2 - t1) \* 1000);

return 0;

}