Convolution

Thomas Saurel

October 25, 2025

Contents

1 PNG file format

This program uses the **PNG (Portable Network Graphics)** extension as input because it is one of the most common formats used to save images; it supports transparency and uses lossless compression.

A PNG file is composed of at least:

1. A **signature** : $(137\ 80\ 78\ 71\ 13\ 10\ 26\ 10)_2$ or $(89\ 50\ 4E\ 47\ 0D\ 0A\ 1A\ 0A)_{16}$
2. An **IHDR** chunk : width, height, bit depth, color type, compression method, filter method, interlace method
3. An **IDAT** chunk : data (can appear multiple times)
4. An **IEND** chunk : end of the data

Each chunk is composed of :

1. **Length** : 4 bytes

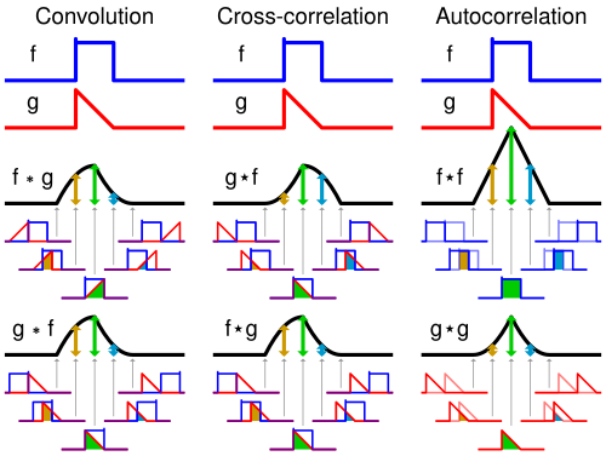
2. **Type** : 4 bytes
3. **Data** : variable
4. **CRC** : 4 bytes

I used the libpng (png.h) library to implement the algorithm in C. You can find the manual here.
This program does not work if the extension of the file was improperly changed to png, such as renaming a jpg file to png, as the header will not be correctly set up.

2 Discrete convolution

The convolution of two real-valued functions f and g is defined as:

$$(f * g)(x) := \int_R f(x-t)g(t)\,dt = \int_R f(t)g(x-t)\,dt$$



Example of convolutions (from Wikipedia)

In the context of image convolution, it takes an original image and a **kernel/filter**. A kernel is a small matrix (I mostly used 3x3, 5x5, and 7x7 matrices) that describes the modifications made to the original image. It must be odd-sized to center it on the desired pixel.

For example, the **Gaussian Blur 5x5** kernel:

$$K_{GB_5} = \frac{1}{273}\begin{pmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{pmatrix}$$

If we define:

1. A kernel of size $(2m+1) \times (2n+1)$,
2. $I(x, y)$ the value of a pixel (R,G,B or A) at the coordinates (x, y),
3. $K(i, j)$ the value of the kernel at the (i, j) coordinates,

we get the equality:

$$P(x, y) := \sum_{i=-m}^{m} \sum_{j=-n}^{n} I(x+i, y+j)K(i, j)$$

where $P(x, y)$ is the new value of the pixel (x, y) after convolution. Intuitively, the sums run through every pixel neighbour of the (x, y) pixel. Each neighbour is multiplied by the coefficient inside the kernel. Then we add every result from the neighbour to get the new value of the pixel. We then repeat the process on each of the four channels to get an $RGBA$ value.

3 C implementation

You first need to enter the command **make** in a terminal in the folder that contains the Makefile. To use the program, use the following command:

```
./main INPUT_LOCATION OUTPUT_LOCATION KERNEL_ID
```

KERNEL_ID represents the kernel that you want to use:

1. IDENTITY : 0
2. BLUR 3x3 : 1
3. BLUR 5x5 : 2
4. BLUR 7x7 : 3
5. GAUSSIAN BLUR 3x3 : 4
6. GAUSSIAN BLUR 5x5 : 5
7. GAUSSIAN BLUR 7x7 : 6
8. SHARP : 7
9. ULTRA SHARP : 8
10. SOBEL X : 9
11. SOBEL Y : 10
12. LAPLACIAN : 11
13. EMBOSS : 12
14. OUTLINE : 13
15. CHECKER 3x3 : 14
16. CHECKER 9x9 : 15

Of course, you can define your own kernel, for example:

```
float MY_KERNEL[9] = {
    -2.f, -1.f, 0.f,
    -1.f, 1.f, 1.f,
    0.f, 1.f, 2.f
};
```

I did not find any issues or memory leaks (checked with Valgrind). However, some kernels may not work well on certain images. For example, the outline kernel is sensitive to contrast: if the image has low contrast or lacks a transparent border, the kernel might produce near-zero results. It works best with images that have high contrast or a translucent background.

4 Kernels

Here is a list of all default kernels implemented.

1. Identity : $\begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$: Does not change the image

2. Blur : $\frac{1}{9}\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}, \frac{1}{25}\begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix}, \frac{1}{49}\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$

   Averaging blur
   $\sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij} = 1 \implies$ Keep the same average intensity

3. Gaussian Blur : $\frac{1}{16}\begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}, \frac{1}{273}\begin{pmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{pmatrix}, \frac{1}{1003}\begin{pmatrix} 0 & 0 & 1 & 3 & 1 & 0 & 0 \\ 0 & 3 & 13 & 22 & 13 & 3 & 0 \\ 1 & 13 & 59 & 97 & 59 & 13 & 1 \\ 2 & 22 & 97 & 159 & 97 & 22 & 2 \\ 1 & 13 & 59 & 97 & 59 & 13 & 1 \\ 0 & 3 & 13 & 22 & 13 & 3 & 0 \\ 0 & 0 & 1 & 3 & 1 & 0 & 0 \end{pmatrix}$

   Smooth image with Gaussian weights

   Weights are given by the gaussian function $G(x, y) = \frac{1}{2\pi\sigma}e^{-\frac{x^2+y^2}{2\sigma}}$

4. Sharp : $\begin{pmatrix} 0 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 0 \end{pmatrix}$: Enhances edges slightly

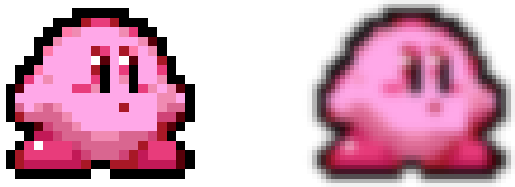
5. Ultra Sharp : $\begin{pmatrix} 0 & 0 & 0 \\ 0 & 10 & 0 \\ 0 & 0 & 0 \end{pmatrix}$: Stronger edge enhancement

6. Sobel X : $\begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$: Edge detection (left) in horizontal direction

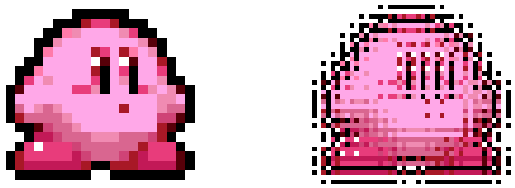
7. Sobel Y : $\begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$: Edge detection (top) in vertical direction
8. Laplacian : $\begin{pmatrix} -1 & 2 & -1 \\ 2 & -4 & 2 \\ -1 & 2 & -1 \end{pmatrix}$: Detects edges in all directions
9. Emboss : $\begin{pmatrix} -2 & -1 & 0 \\ -1 & 1 & 1 \\ 0 & 1 & 2 \end{pmatrix}$: Creates a 3D emboss effect
10. Outline : $\begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix}$: Highlights strong edges
11. Checker : $\begin{pmatrix} 1 & -1 & 1 \\ -1 & 1 & -1 \\ 1 & -1 & 1 \end{pmatrix}, \frac{1}{9} \begin{pmatrix} 1 & 1 & 1 & -1 & -1 & -1 & 1 & 1 & 1 \\ 1 & 1 & 1 & -1 & -1 & -1 & 1 & 1 & 1 \\ 1 & 1 & 1 & -1 & -1 & -1 & 1 & 1 & 1 \\ -1 & -1 & -1 & 1 & 1 & 1 & -1 & -1 & -1 \\ -1 & -1 & -1 & 1 & 1 & 1 & -1 & -1 & -1 \\ -1 & -1 & -1 & 1 & 1 & 1 & -1 & -1 & -1 \\ 1 & 1 & 1 & -1 & -1 & -1 & 1 & 1 & 1 \\ 1 & 1 & 1 & -1 & -1 & -1 & 1 & 1 & 1 \\ 1 & 1 & 1 & -1 & -1 & -1 & 1 & 1 & 1 \end{pmatrix}$
    Produce a checkerboard pattern

5 Some results

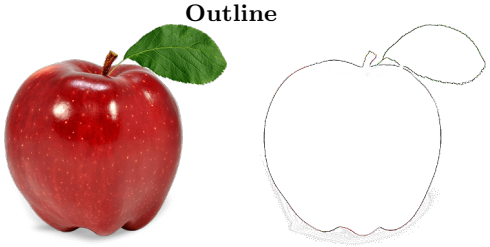
Gaussian Blur



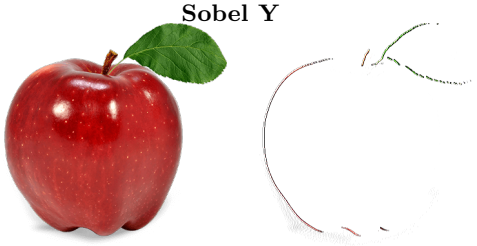
Checker



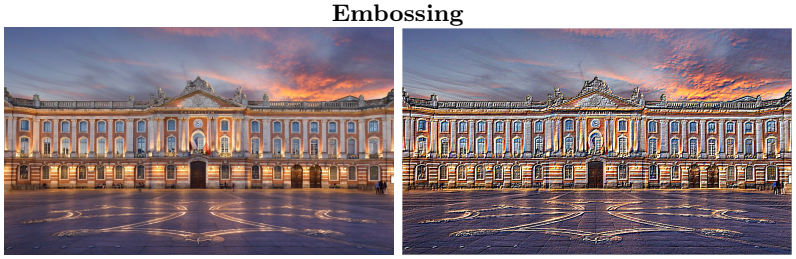
Outline



Sobel Y



Embossing



Bigger Checker

6 Bibliography

Main online references :

1. Wikipedia : PNG
2. Wikipedia : Kernel
3. libpng.org
4. Wikipedia : Convolution
5. Various discussions on Stack Overflow (used for debugging and implementation)