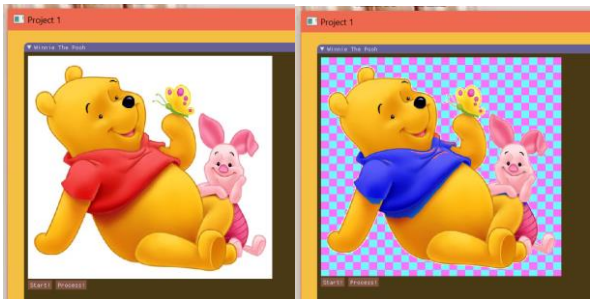


# PROJECT 1

Image load and pixels access

Mila Anastasova

# Video & Image Processing

## Project description and goal

The project requires to use the ImGui libraries to create a program with friendly user interface. The main purpose for the program's functionality is to allow an image loading and a processing it, changing the values of the different components inside the pixels.

## Implementation description

The implementation involves different steps in order to accomplish all the requirements.

First of all, the main window has to be created.

```
GLFWwindow* window = glfwCreateWindow(1000, 700, "Project 1", NULL, NULL);
glfwMakeContextCurrent(window);
```

A constructor for the window is created containing all its parameters as an input value. The first two parameters correspond to the width and the height of the desired window. As third parameter we insert the name of the main window. After all the necessary parameters are defined, a function that creates the window with the inserted values is invoked. A color for the background of the main window is set using a special vector structure.

An infinite "while" loop is created so that the program runs until the user closes it and forces the program to stop. Inside the loop the characteristics of the next window – the window created in the next iteration of the loop. An inner window is created. The user can change the size of it, mi minimize it, and to press all the buttons. All these actions are immediately provoking the change of the window and in the next iteration the newly created window is different.

For the first functionality of the program – load an image from the database in the computer and show it on the screen, a button named "Start!" is created. When pressing the button an image called "pooh1.jpg" is loaded into the structure "a" of Image type. The function `stbi_load("name.jpg", &image.width, &image.height, &image.n, RGBA)` automatically puts all the information of the loaded image inside the attributes of the Image. The width, the height, the number of components per pixel and order of the components are accessible from now.

To show the content of the image that is now loaded in the memory and is also in the Image structure "a", a function `texture_image(&a)` is called passing pointer to the first pixel of the image.

This ends the implementation of the first part of the project and the functionality of the first button.

The second functionality requires change of the components of the pixels. This part is not restrictive, and any algorithm can be applied in order to create a new image or to change the features of the loaded one.

Pressing a button called "Process!" is placed next to the button "Start!". Once this button is pressed the image content is changed and the changes are immediately displayed on the screen.

The algorithm used for this program involves change of the background and of particular parts of the image.

First of all, all the image has to be traverse. As the image inside the memory is just a sequence of bits, but for the users it is a matrix – has rows and columns, all the loops allow the usage of only one counting variable but also allow the implementation using two counters – one for the rows and one for the columns.

Some auxiliary variables are declared at the beginning – “inBlue” is a boolean variable that indicates if a given section, currently accessed, is blue or not blue – purple. A variable count is used to count the number of passed rows, so that later we can make changes when some certain number of rows is reached. The number of bytes per pixel is stored in a variable called “`bytesPerPixel`” and the number of bytes per row is also stored in a variable called “`bytesPerRow`”.

The first “for” loop is passing through all the rows. This means the all the position of a row will be accessed before going to the next row. This implementation requires another “for” loop nested to this one so that all the columns can be accessed in each row. However, the number of the passed rows is checked. If the numbers of rows passed is twenty, the counter is set again to zero and the ”inBlue” variable is set to its opposite. This allows to change the starting color for each 20 rows – 20 rows will start being blue, then 20 will start being purple and so on.

The second loop is accessing each 20$^{th}$ pixel of each row. This is how the usage of counter can be avoided. The check of the “inBlue” is made again so that the columns are alternating as well. The combination of both loops and the conditions for the color and the counter of the rows and columns allow to make a chess board pattern.

In order to change only some part of the image – only the background, the values of each component of a pixel are checked. If all of them are higher than 250, the color is changed. To access all the pixels of a column, another “for” loop is created going only from 0 to 20. Then the components we want to access are reached by `a.pixels[((row)* bytesPerRow + (col * bytesPerPixel)) + (0;1;2;3) + i * 4].` To pass all the image that has been already processed, first the number of rows passed is multiplied by the number of bytes per row. The number of columns passed is multiplied by the bytes per pixel and both are added. The column for loop is skipping 20 rows and another loop using i is implemented, so the value of i*bytesPerPixel has to be added. That is the approach to get to every pixel of the image.

Nevertheless, every pixel is composed of 4 components – 4 bytes. To reach all of them, a number between 0 and 3 is added, where 0 is for the red component byte, 1 is for the green, 2 is for the blue and 3 is for the alpha component. Changing the values of the component bytes changes the color of the pixel.

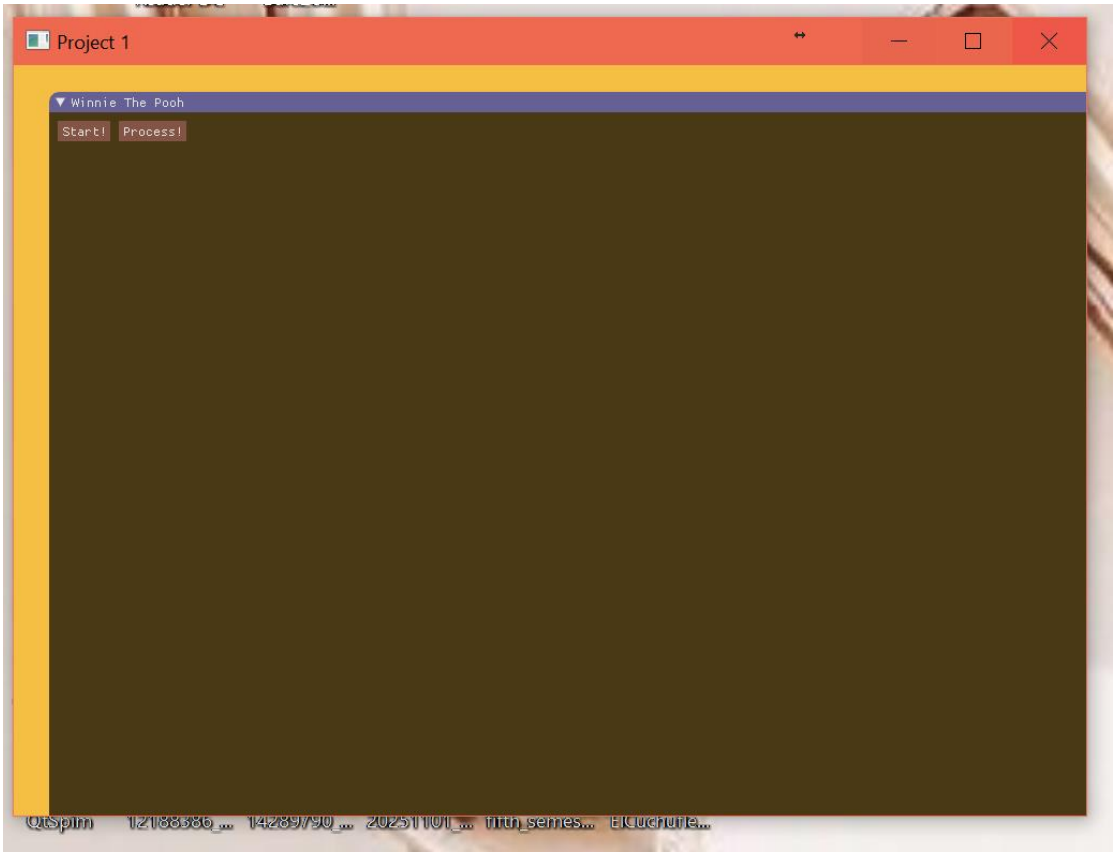
Besides changing the background, a particular part of the image is modified as well when the “process!” button is pressed. The T-shirt of Winnie the pooh is changes its color.

First of all, the color of the currently accessed pixel is checked and it will be changed only if it is red – the color of the T-shirt. As there may be more red elements in the image, the zone of the change is reduced only from 1/3 to 2.2/3 of the height and from 1/3 to 2.4/3 from the width. Only when we are inside this zone the color red will be modified. If this is the case the red component of the image will be exchanges with the blue using auxiliary variable. In this way the shades of the image are not going to be changed but the color is going to be different.

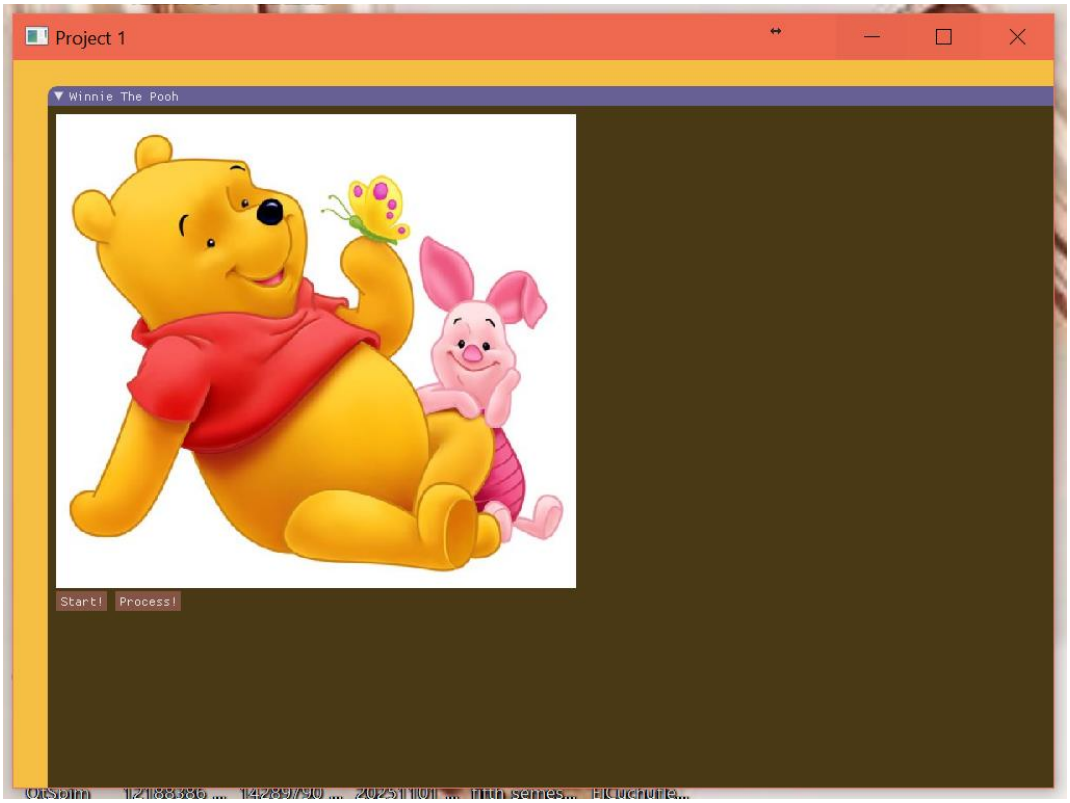
The same algorithm is implemented in the else part of the if statement when the pointer points to part that is going to become purple.

At the end the memory of the image is freed, the window is rendered and in case if exit of the program the window is shut down.

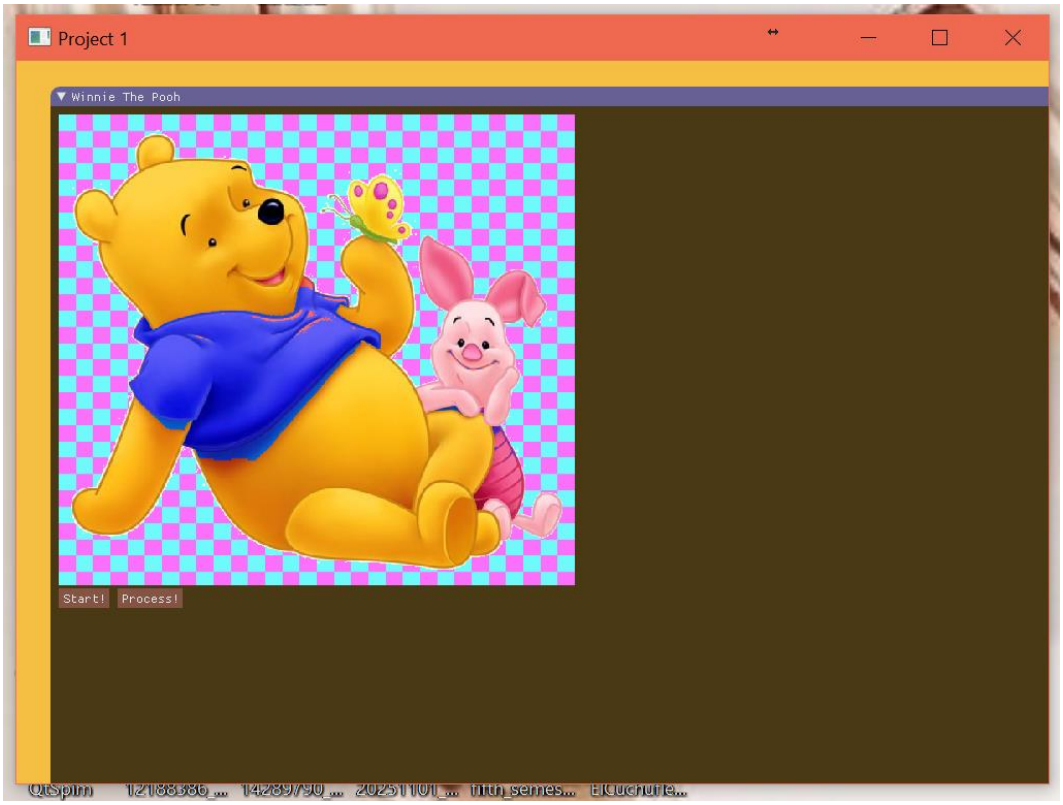
## Gallery




Screenshot of the program main window at the start of the program.




Screenshot when the button “Start!” is pressed.




Screenshot when the button “Process!” is pressed.