M2-BIG DATA
GPGPU Chapter 3

Exercice 3

Réalisé par:
Gaby Maroun

Encadré par:
Dr. Etancelin JM

December 9, 2020

Objectives

The purpose is to implement an image blurring on GPU. We use a 3x3 box filter to blur the input image according to the following formula :

$$C_{i,j} = 1/9 \sum_{k=-1}^{1} \sum_{l=-1}^{1} C_{i+k,j+l}$$

where $C_{i,j}$ is the value of the channel C of the image at row i and column j.

Instructions

Edit the given code to perform the following :

```
#include <iostream>
#include <cstdlib>
#include <string>


#include "img_utils.hpp"


#define BLUR_SIZE 10
using namespace std;

//@TODO@ : Write the kernel here

__global__ void blurKernel(float * image, float * blurImage, int w, int h, int
    channels) {
  int Col  = blockIdx.x* blockDim.x+ threadIdx.x;
  int Row  = blockIdx.y* blockDim.y+ threadIdx.y;

  if(Col < w && Row < h) {
    int pixels = 0;
    float r = 0;
    float g = 0;
    float b = 0;
    // Get the average of the surrounding 2xBLUR_SIZE x 2xBLUR_SIZE box
    for(int blurRow = -BLUR_SIZE; blurRow< BLUR_SIZE+1; ++blurRow) {

      for(int blurCol = -BLUR_SIZE; blurCol < BLUR_SIZE+1; ++blurCol) {

        int curRow = Row + blurRow;
        int curCol = Col + blurCol;

        // Verify we have a valid image pixel
        if(curRow > -1 && curRow < h && curCol > -1 && curCol < w) {

          r += image[(curRow * w + curCol) * channels + 0];
```

```
35
36          if (channels == 3){
37            g += image[(curRow * w + curCol) * channels + 1];
38            b += image[(curRow * w + curCol) * channels + 2];
39            }
40
41          pixels++;// Keep track of number of pixels in the accumulated total
42          }
43        }
44      }
45
46    // Write our new pixel value output
47    blurImage[(Row * w + Col) * channels] = (float)(r/ (pixels));
48    if(channels ==3){
49      blurImage[(Row * w + Col) * channels + 1] = (float)(g/ pixels);
50      blurImage[(Row * w + Col) * channels + 2] = (float)(b/ pixels);
51    }
52
53
54  }
55 }
56
57 int main(int argc, char **argv)
58 {
59   if(argc!=3) {
60   cout<<"Program takes two image filenames as parameters"<<endl;
61   exit(3);
62   }
63
64   float *imgIn, *imgOut;
65   int nCols, nRows, channels;
66
67   // Allocate images and initialize from file
68   imgIn = read_image_asfloat(argv[1],&nCols, &nRows, &channels);
69
70   imgOut = (float *)calloc(nCols * nRows * channels, sizeof(float));
71
72   // Allocates device images
73   float *d_imgIn, *d_imgOut;
74
75   //@TODO@ : Complete for device allocations
76   cudaMalloc(&d_imgIn, nCols * nRows * channels * sizeof(float));
77   cudaMalloc(&d_imgOut, nCols * nRows * channels * sizeof(float));
78
79
80   // Copy input data
81   //@TODO@ : Complete for data copy
82   cudaMemcpy(d_imgIn, imgIn, nCols * nRows * channels * sizeof(float),
       cudaMemcpyHostToDevice );
83
84
```

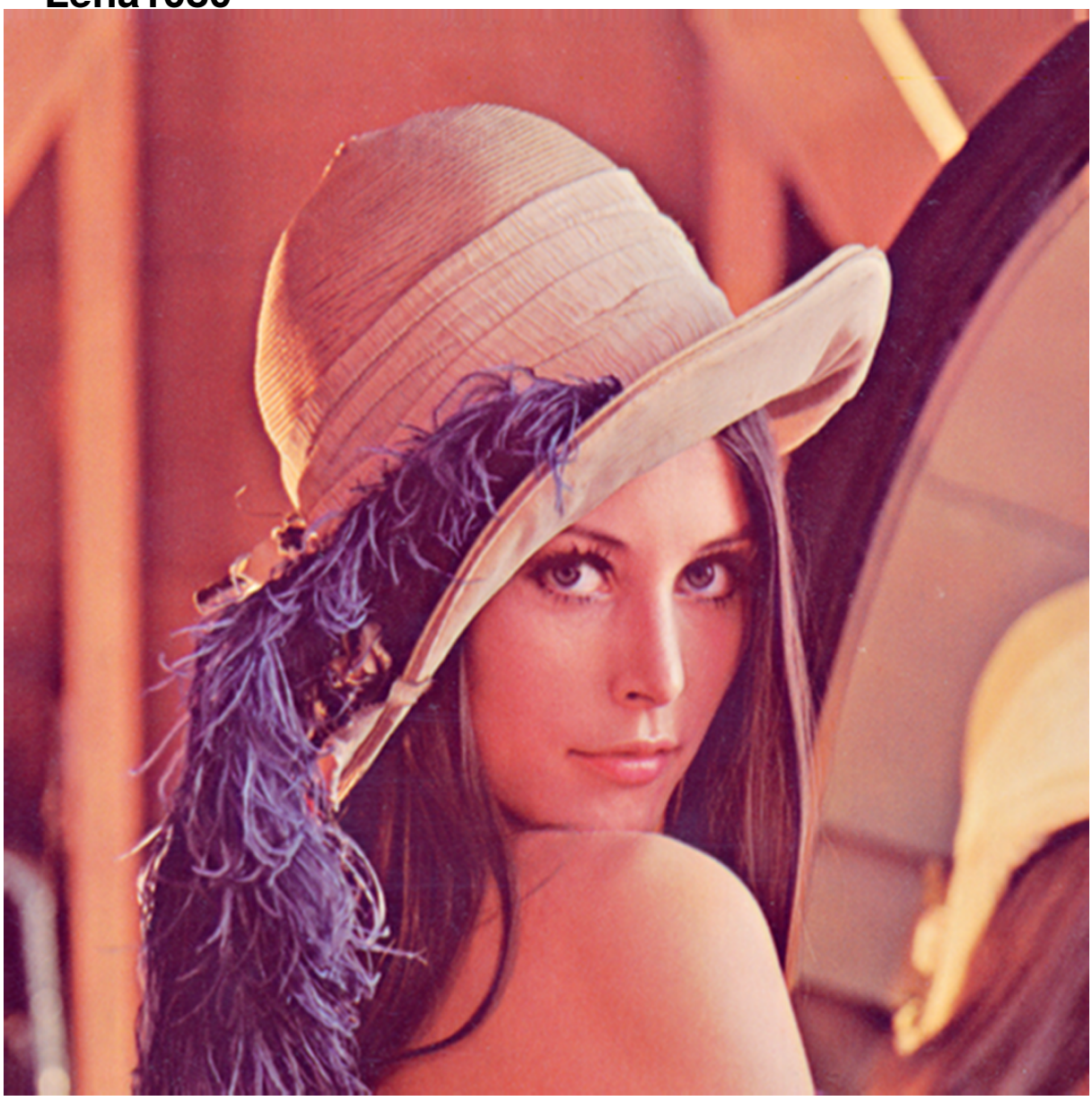
```
85
86    // Call the kernel
87    //@TODO@ : Compute threads block and grid dimensions
88    //@TODO@ : Call the CUDA kernel
89    dim3 DimGrid((nCols-1)/32+ 1, (nRows-1)/32+1, 1);
90    dim3 DimBlock(32, 32, 1);
91
92    blurKernel<<<DimGrid,DimBlock>>>(d_imgIn, d_imgOut, nCols, nRows, channels);
93
94
95    // Copy output data
96    //@TODO@ : Complete for data copy
97    cudaMemcpy(imgOut, d_imgOut, nCols * nRows * channels * sizeof(float),
        cudaMemcpyDeviceToHost );
98
99    // Write gray image to file
100   write_image_fromfloat(argv[2], imgOut, nCols, nRows, channels);
101
102   // Free memory
103   //@TODO@ : Free host and device memory
104   // Check result
105   // check(host_c,n);
106
107   // Free device memory
108   cudaFree(d_imgIn);
109   cudaFree(d_imgOut);
110
111   // Free host memory
112   free(imgIn);
113   free(imgOut);
114
115   return 0;
116 }
```

NB: In the last deliverable, I was told in the comments section by your side that the grids are not consistent. I replaced the first parameter of the $dimGrid$ function with the second. I wish that was what you meant.

The resulted images came as follow :

Lena

Lena1080



Lena B&W

Questions

1. **How many floating operations are being performed in your blurring kernel ? EXPLAIN.**

   There are $nCols*nRows*channels(1+((2*BLURSIZE)+1)^2)$ *floating operations being performed depending on the number of channels of the entered image for each division over the number of pixels plus the number equal to the BLURSIZE matrix size of addition performed to the floating variable.*
   The length of a row or a column in the matrix is equal to $2 * BLURSIZE$ *going from* $-BLURSIZE$ *to* $BLURISZE$ *and the +1 represents the pixel itself.*

   (a) *If the image is of 1 color, the program face 1 division and* $((2 * BLURSIZE) + 1)^2$ *addition in the for loops that means* $nCols * nRows * 1(1 + ((2 * BLURSIZE) + 1)^2)$.

   (b) *While, if the program encounter 3 colors, the program would have to do 3 division and the additions for each channel so* $nCols * nRows * 3(1 + ((2 * BLURSIZE) + 1)^2)$.

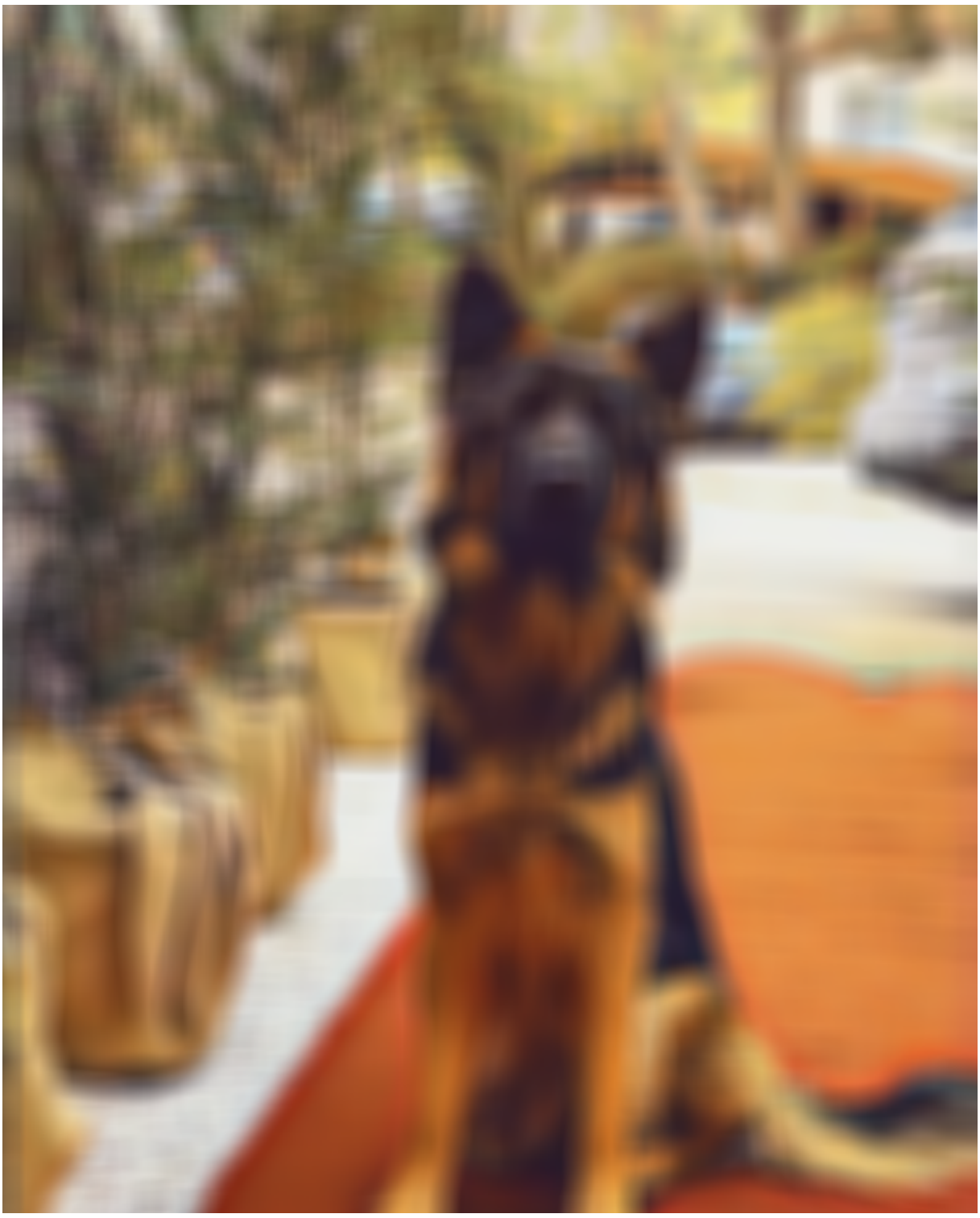
2. **How many global memory reads are being performed by your kernel ? EXPLAIN.**
   It's equal to $nCols * nRows * channels * ((2 * BLURSIZE) + 1)^2$ *to read the whole image with its different channels of colors as the condition in the* if *is to search the whole width and height and each time enter and read from the* for *loops for* $((2 * BLURSIZE) + 1)^2$ *times so it's depending on the type of input image the program is facing and the size of the blurring matrix.*

3. **How many global memory writes are being performed by your kernel ? EXPLAIN.**
   It's equal to $nCols * nRows * channels$ *to write the whole new image using the number of channels entered to perform as much writes in the outputed image and blur the content*

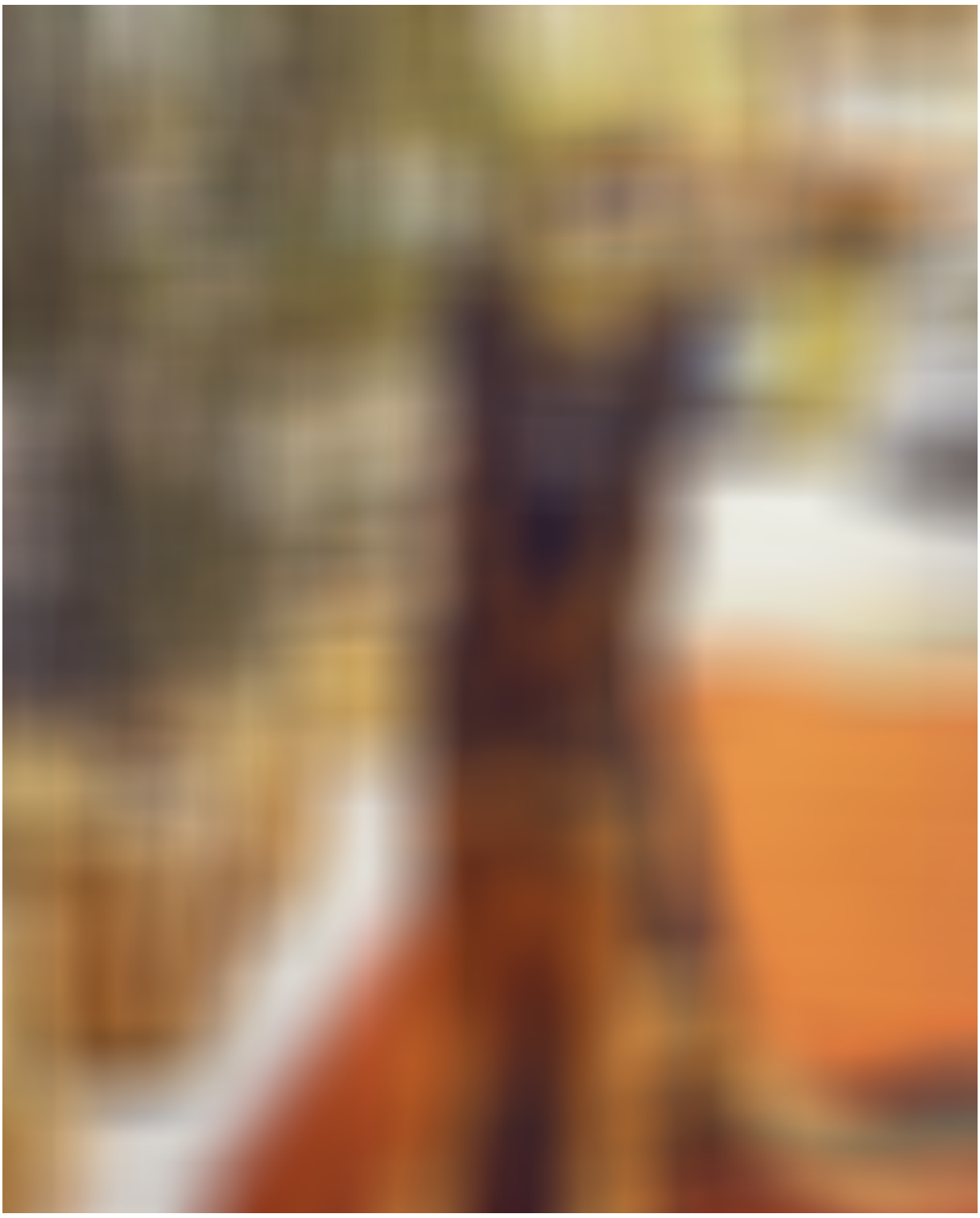
4. **What could you intend to do in order to have a stronger blur effect ? Try in your kernel, and compare the results.**
   I have found 3 ways to blur in different and sometimes stronger ways:

   (a) Decreasing the BLURSIZE or the curRow and the curCol value:
   For example: BLURSIZE = 30 gives the following picture,

Remember Ivy ?!

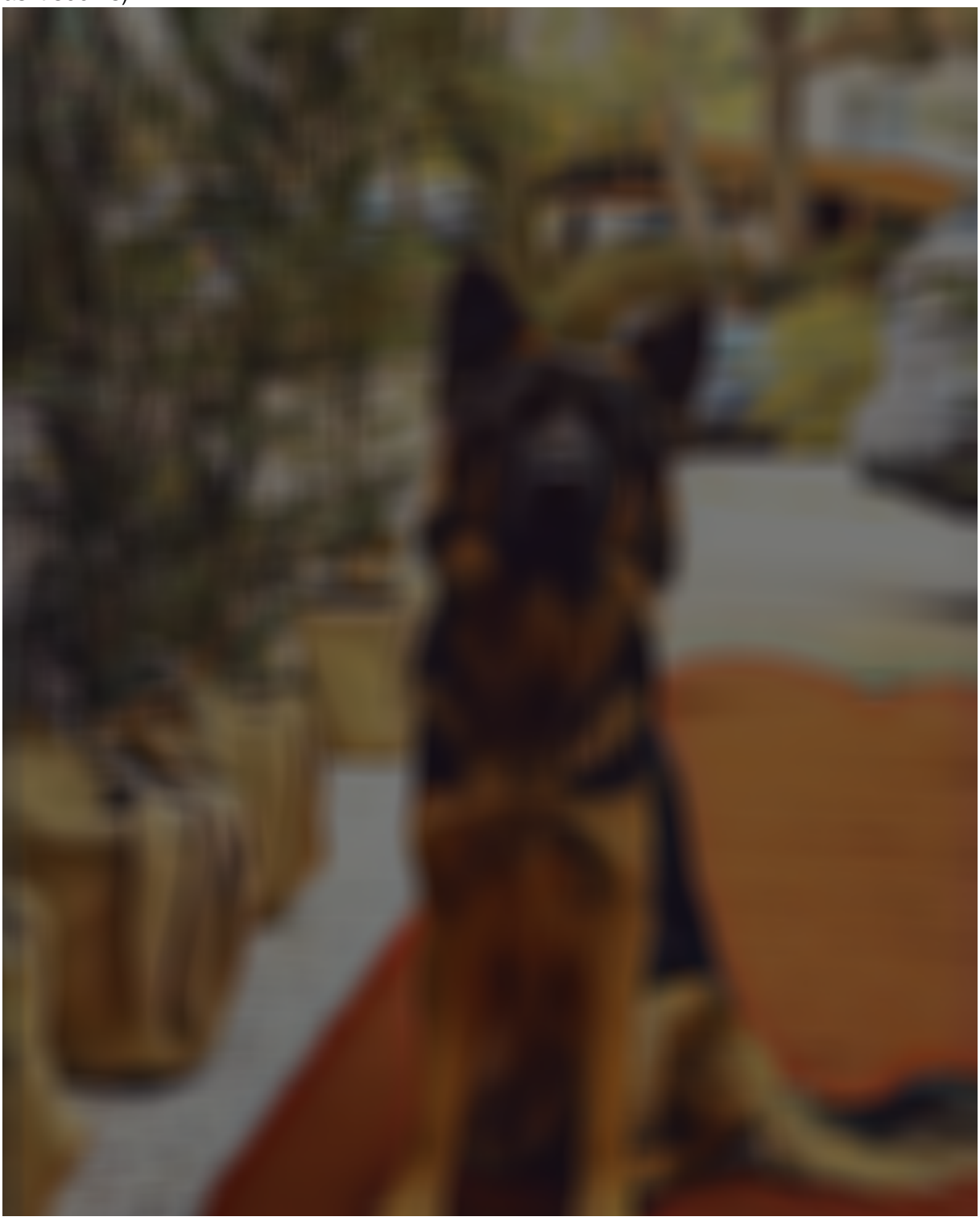


Ivy BlurSize = 10



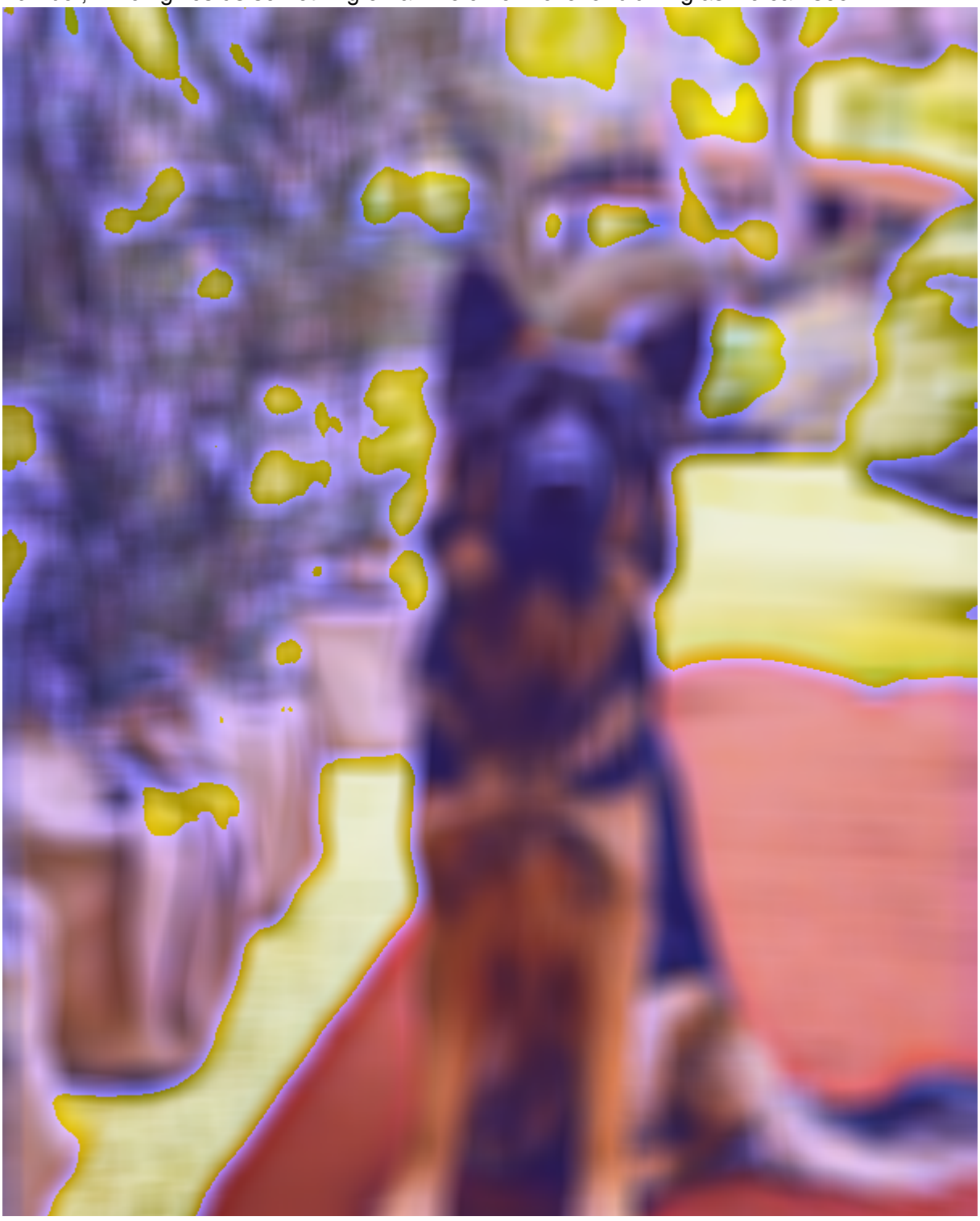
Ivy BlurSize = 30

(b) Multiplying the pixels variable by any > 1 number (might affect the brightness also as it seems)



Ivy pixels * 2

(c) Multiplying the pixels variable by any 0 <number< 1 or the r, g & b with any > 1 number, which gives us something on a whole new level of blurring as we can see.



Ivy pixels * 1/2

La fin.