# HPC_HW4:Report

Open the directory "HW_cuda" on the github.

# Temperature distribution

Laplace equation solution is written on the file "cuda_laplace.cu".

## Kernels

Main kernels are the following:

```
// CUDA KERNEL FUNCTIONS


__global__ void Initialization(int N, int N_2d, float *d_a)
{
    //  n = N * N

    int globalidx = threadIdx.z * blockDim.x * blockDim.y + threadIdx.y * blockDim.x + threadIdx.x;
    if(globalidx<N)
  {
      if (globalidx >= 1 && globalidx <= N_2d-2)
```

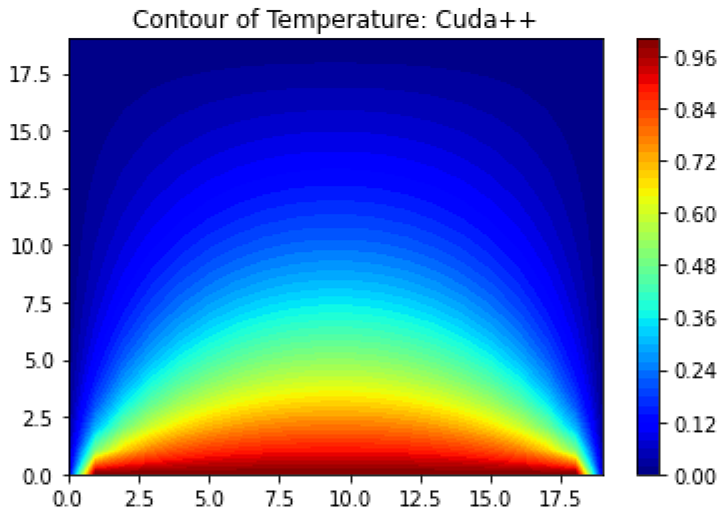
```
   {
     printf("Hello from tx = %d\t ty=%d\t tz=%d\t gi=%d\n", threadIdx.x, threadIdx.y, threadIdx.z, globalidx);
     d_a[globalidx]=1.0;
   } else
   {
     d_a[globalidx]=0.0;
   }

 }
 printf("from tx = %d\t ty=%d\t tz=%d\t gi=%d\n", threadIdx.x, threadIdx.y, threadIdx.z, globalidx);

}

__global__ void Laplace(int N, float *T, float *d_res)
{
  int globalidx = threadIdx.z * blockDim.x * blockDim.y + threadIdx.y * blockDim.x + threadIdx.x;
  if(globalidx<N*N)
  {
    if ((globalidx >= 0 && globalidx <= N-1) | (globalidx % N == 0) | (globalidx >= N*N-N) | (globalidx % N >= N -1 && globalidx % N <= N*N -1))
    {
      //printf("Hello from tx = %d\t ty=%d\t tz=%d\t gi=%d\n", threadIdx.x, threadIdx.y, threadIdx.z, globalidx);
      d_res[globalidx]=T[globalidx];

    } else
    {
      int top, bottom, left, right;
      top = -N + globalidx;
      bottom = N + globalidx;
      left = -1 + globalidx;
      right = 1 + globalidx;
      d_res[globalidx]=0.25 * (T[top] + T[bottom] + T[left] + T[right]);
    }

    }


}
```

## Compilation

To compile it do the following:

```
!nvcc -arch=compute_50 cuda_laplace.cu -o hello && ./hello
```

## Results



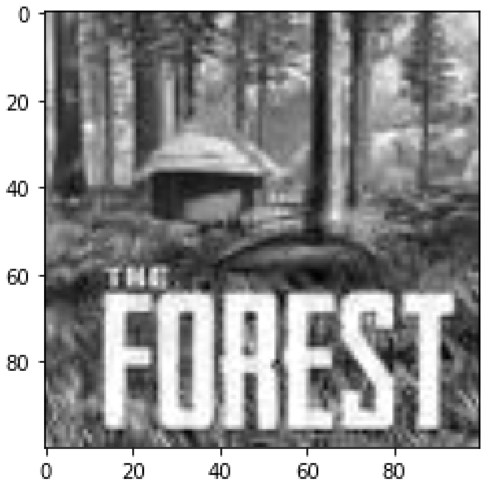
# Blurring and Cartoon filter

## Original Image

I uploaded the image file from my desktop by:

```
uploaded = files.upload()
```

Before run this code, you should download this picture "**HPC_HW_4_picture_100_100.jpg**" on your desktop, and then run it, choosing this picture.
Picture looks like this:



Then to take an array, image was converted in the following way:

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import imageio
import os
from google.colab import files
import sys
from numpy import asarray
from IPython import display
from PIL import Image
import PIL


def image_import():
    image = Image.open('HPC_HW_4_picture_100_100.jpg')
    data = asarray(image)
    im_data = data[:,:,0]
    return im_data
```

```
x = image_import()
```

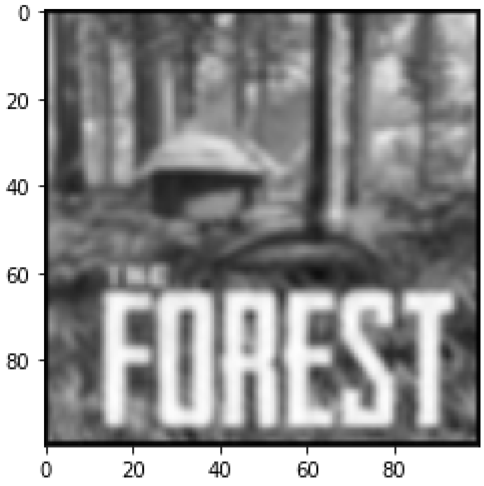
And to save array into txt file, do the following:

```
import numpy as np
```

```
np.savetxt("image_array.txt", np.array(x), fmt="%s")
```

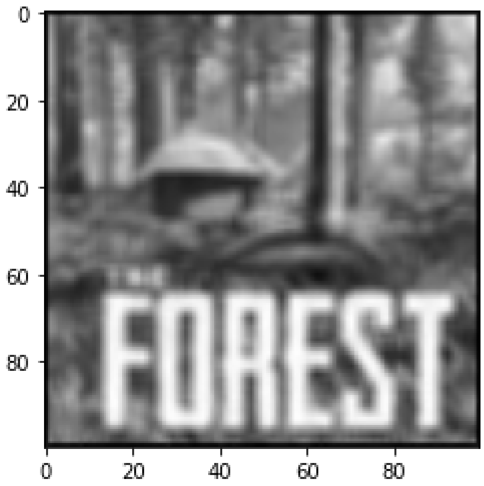
## Blured Image 3x3 kernel Gaussian blur

```
kernel = [[1,2,1],
          [2,4,2],
          [1,2,1]]
```



## Blured Image 3x3 Box blur

```
kernel = [[1,1,1],
          [1,1,1],
          [1,1,1]]
```



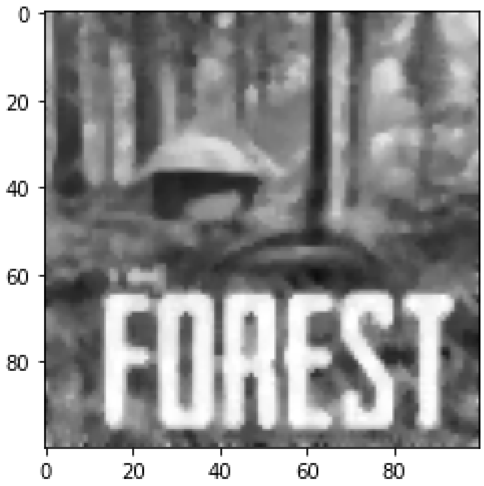
## Blured Image with median filter

Here I used bubble sort of neighbours and took the median value of sorted array.

```
__global__ void Bluring_median_filter(int array_2D_size, int *array_1D_cuda, int *blured_1D_cuda, int *neighbours)
{
    int globalidx = blockIdx.x * blockDim.x + threadIdx.x;
    int N = array_2D_size;


    if(globalidx<N*N)
    {
        if ((globalidx >= 0 && globalidx <= N-1) | (globalidx % N == 0) | (globalidx >= N*N-N) | (globalidx % N >= N -1 && globalidx % N <= N*N -1))
        {
            blured_1D_cuda[globalidx]=array_1D_cuda[globalidx];
        } else
        {

            int top, bottom, left, right;
            top = -N + globalidx; // [i-1][j]
            bottom = N + globalidx; // [i+1][j]
            left = -1 + globalidx; // [i][j-1]
            right = 1 + globalidx; // [i][j+1]
```

```

      int cross1, cross2,cross3,cross4;
      cross1 = globalidx - N - 1; // [i-1][j-1]
      cross2 = globalidx - N + 1; // [i-1][j+1]
      cross3 = globalidx + N - 1;  // [i+1][j-1]
      cross4 = globalidx + N + 1;  // [i+1][j+1]

      neighbours[0] = array_1D_cuda[top];
      neighbours[1] = array_1D_cuda[bottom];
      neighbours[2] = array_1D_cuda[left];
      neighbours[3] = array_1D_cuda[right];
      neighbours[4] = array_1D_cuda[cross1];
      neighbours[5] = array_1D_cuda[cross2];
      neighbours[6] = array_1D_cuda[cross3];
      neighbours[7] = array_1D_cuda[cross4];
      neighbours[8] = array_1D_cuda[globalidx];

      // Сортировка массива пузырьком
      int size = 9;
      for (int i = 0; i < size - 1; i++)
      {
       for (int j = (size - 1); j > i; j--) // для всех элементов после i-ого
       {
        if (neighbours[j - 1] > neighbours[j]) // если текущий элемент меньше предыдущего
        {
         int temp = neighbours[j - 1]; // меняем их местами
         neighbours[j - 1] = neighbours[j];
         neighbours[j] = temp;
        }
       }
      }
      blured_1D_cuda[globalidx] = neighbours[4];
    }
  }
```

The result is the following:



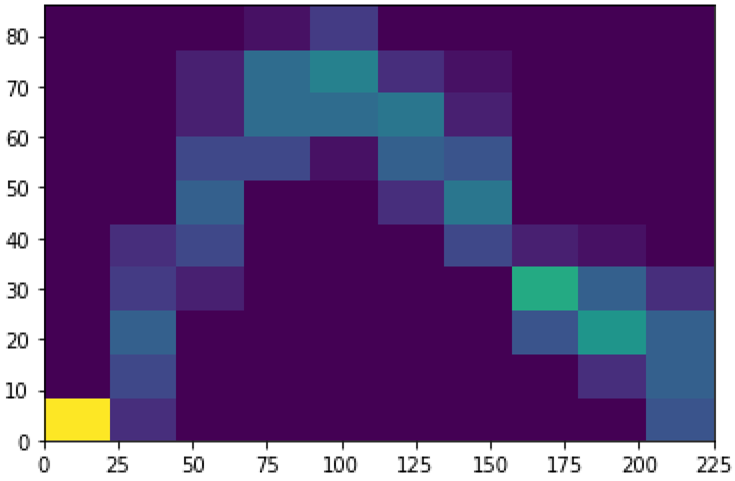
## Histogram

To compile the code do the following:

```
!nvcc -arch=compute_50 cuda_histogram.cu -o histogram && ./histogram > histogram_array.txt
```

## Result

```
a = np.fromfile('histogram_array.txt', dtype=float, count=-1, sep=' ')
plt.hist2d([i for i in range(226)],a);
```


80
70
60
50
40
30
20
10
0
0
25
50
75
100
125
150
175
200
225