APPROXIMATE COMPUTING LAB

DESIGN SPACE EXPLORATION

Introduction

This goal of this lab is to introduce the basic knowledge of approximation technique. You will work with an image segmentation application based on k-means clustering algorithm[1]. The application is implemented in C++.

Let's start

The lab directory is organized as follows:

- Src: it contains the source code of the segmentation algorithm
- Image: it contains the input image (file format png)
- Script: it contains the python script to convert file forma (from png to rgb and viceversa)

Compile from command line:

```
g++ -std=c++11 ./src/segmentation.cpp ./src/main.cpp -I./src -o segmentation
```

The program can manipulate images in RGB format[2]. We can convert the input image using the provided python script:

```
python3 ./script/png2rgb.py rgb ./image/original.png ./image/input.rgb
```

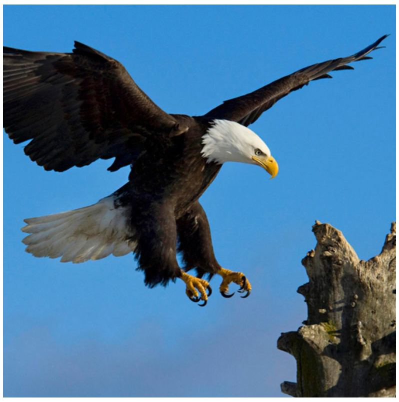

Figure 1: original.png

[1] https://www.sciencedirect.com/science/article/pii/S1877050915014143
[2] https://en.wikipedia.org/wiki/Silicon_Graphics_Image

We can now run the program as follows:

```
./segmentation ./image/input.rgb ./image/outgolden.rgb
```

You should see the following message

```
Error = 0
```

To check the output image, we have to convert back the `outgolden.rgb` into png file

```
python3 ./script/png2rgb.py png ./image/outgolden.rgb ./image/outgolden.png
```

Figure 2: outgolden.png

To Do

The segmentation works with the IEEE 32 bit floating point data type. You can easily modify the data precision by specifying the number of bits for the exponent and for the mantissa.

```
int Exp = 8;
int Mant = 23;
```

The goal is implement a simple design space exploration to identify the pareto frontier solution (Error, Gain). The error has to be computed by the Mean Square Error (MSE) implemented in the function:

```
double get_error (RgbImage* golden, RgbImage* axc);
```

The gain has to be computed as the memory foot print reduction. The memory occupation can be simply computed as

```
MF = W x H x #bit
```

Where W and H are the width and height of the image while the #bit is the amount of bit to store each pixel (E+M+1). You have to write the code to compute it.