194588 Chukwuma Fumnanya Divine

Nearest neighbour interpolation and Bi-linear interpolation are two of the interpolation techniques used in image re-sampling. Nearest neighbour applies a scaling factor on the pixels of the original image and derives the pixels of the resampled image (new image), however nearest neighbour interpolation has its flaws whereas it introduces what is called aliasing into the re-sampled image which is a common problem when using nearest neighbour interpolation for re-sampling images.

Bi-linear on the other hand is another interpolation technique which is said to be slightly better than nearest neighbour because it works in 2 directions on a particular pixel during the process of re-sampling and it also uses the 4 surrounding pixels around the target pixel during the process. It doesn’t completely eliminate our aliasing problem but when the output images of both techniques used are compared we can see that the output image from bi-linear interpolation looks better than the output from nearest neighbour interpolation because for the nearest neighbour interpolation we can see rough edges and Bi-linear interpolation smoothens out those edges to a certain degree giving our image a slightly better look than the latter.

We shall be seeing some example image pairs below and then the source codes for both interpolation techniques:

Nearest Neighbour Interpolation source code:

```
#include "Image.h"
#include "math.h"
#include <windows.h>

int main() {

	GrayscaleImage inputimg, outputimg(300, 300);

	inputimg.Load("images/input.png");

	float scale_x = (float)inputimg.GetWidth() / outputimg.GetWidth();

	for (int y = 0; y < outputimg.GetHeight(); y++){
		for (int x = 0; x < outputimg.GetWidth(); x++){
			outputimg(x, y) = inputimg(x * scale_x, y * scale_x);

		}
	}

	outputimg.Save("images/output.png");


	return 0;
}
```

Bi-linear Interpolation source code:

```
#include "Image.h"
#include <math.h>
#include <stdio.h>



int main() {
	//Code for user to be able to input preferred image size.
	int img_width, img_height;
	printf("User!!!, Please kindly enter your desired image size:=> ");
	scanf("%d%d", &img_width, &img_height);
	GrayscaleImage Input;
	Input.Load("images/input.png");
	// code for re-sampled image
	GrayscaleImage Output(img_width, img_height);
	float scl_x = Input.GetWidth() / (float)Output.GetWidth();
	float scl_y = Input.GetHeight() / (float)Output.GetHeight();
	for (int y = 0; y < Output.GetHeight(); y++)
	{
		for (int x = 0; x < Output.GetWidth(); x++)
		{

			int c = floor(x * scl_x);
			int d = floor(y * scl_y);
			float a = (x * scl_x) - c;
			float b = (y * scl_y) - d;
			float img_output = (1 - a) * (1 - b) * Input(c, d) + (1 - a) 
* b * Input(c, d + 1) + a * (1 - b) * Input(c + 1, d) + a * b * Input(c + 1, d + 
1);
			int output = round(img_output);
			if (output > 255)
				output = 255;
			if (output < 0)
				output = 0;
			Output(x,y) = output;
		}
	}
	Output.Save("images/output.png");
}
```

SUMMARY OF HOW BI-LINEAR INTERPOLATION CODE WORKS.

For my Bi-linear interpolation code I had to create 2 variables using an “#include<stdio.h>” library and “scanf” and “printf” functions to be able to specify in my code during the building process where one would be able to input he’s or her desired image size. And then as usual because we are using the “Grayscaleimageinput” function to specify that the image we are using as our input image is supposed to be a grayscale image and not coloured and then we specify our function that retrieves our input image using “Input.load” and the file path where our “PNG” file type image has been stored on our computer (NOTE: only PNG image type can be used in this code) and then we have to get our scale factor that will be used for our image re-sampling and we get them for both axis “scl_x and scl_y” we have the “float” because during the building process we might have ~integer numbers which might not be good for our cause and in some cases the result desired might not be attained and then we have the remaining part of the code which entails the formula used and for the re-sampling/Bi-linear interpolation technique used on our image and also we have our “Output.save” plus the directory to where the new image is going to be saved.

## Image pairs.

Image set 1:

Original Image:

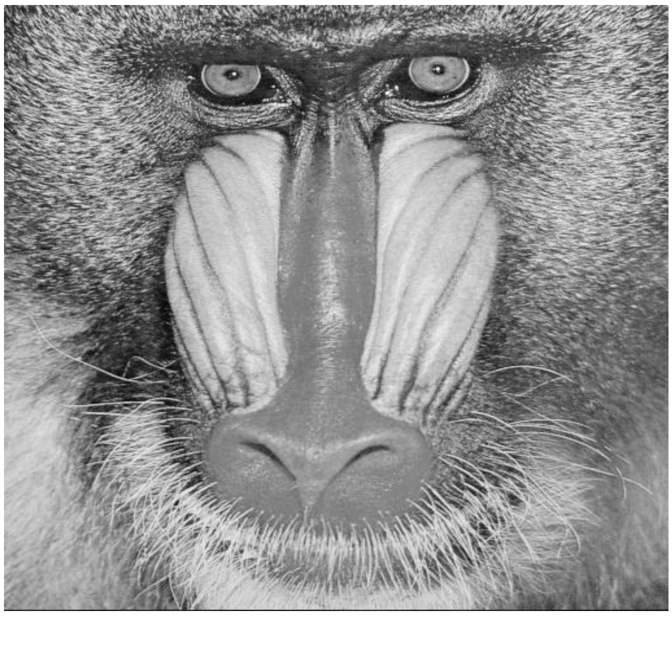
Using Nearest neighbour interpolation:

Using Bi-linear interpolation:

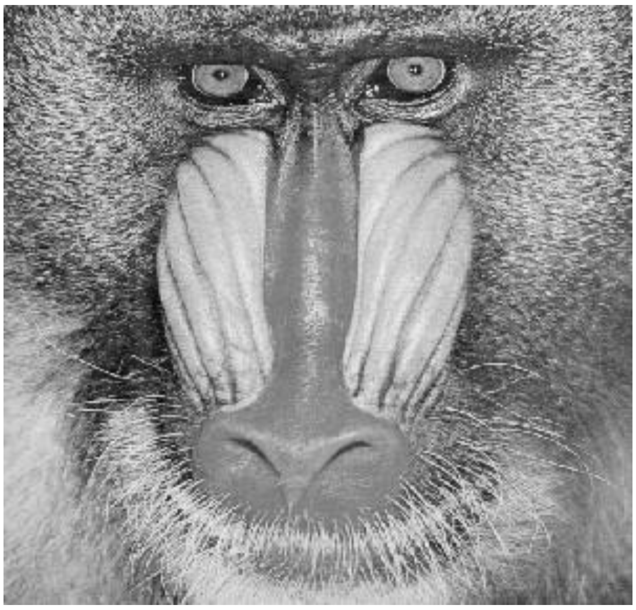
This is image was scaled down from a (x_axis = 512 by y_axis = 512) to a (x_axis = 300 by y_axis = 300) and as we can see from the output images of both techniques aliasing is noticed more in the nearest neighbour re-sampled image at the edges of lena's hat than that of her hat in the bi-linear re-sampled image we can see that those sharp edges have been smoothened out but aliasing has not necessarily been eliminated.

Image set 2:

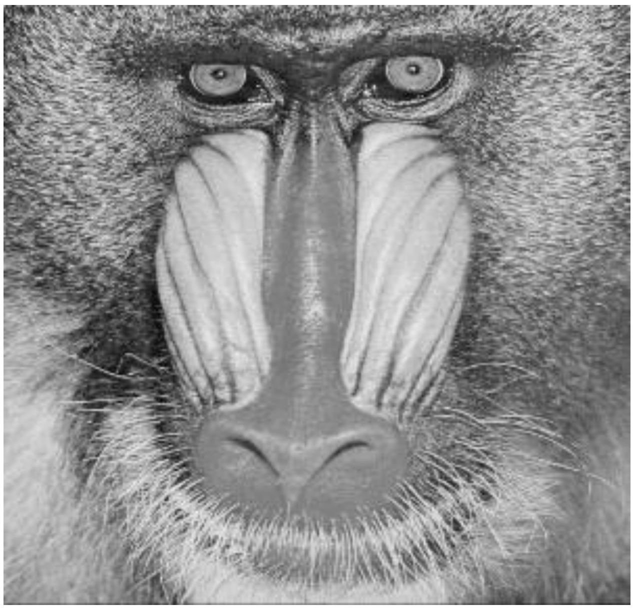
Original image:



Using Nearest neighbour interpolation:



Using Bi-linear interpolation:



In this image we scaled down from an image of dimension (x_axis  = 512 by y_axis = 512) to an image of (x_axis = 300 by y_axis = 300) and we can notice that the re-sampled nearest neighbour interpolation image that we lost some data around the whiskers of the monkey and also the bridge of the nose of the monkey but this is made up for in the Bi-linear interpolation re-sampled image where we can see that it's nowhere near the original image but the amount of data lost in the nearest neighbour has been recovered in the bi-linear to a certain degree.