Student Name : Wei Li
Assignment 2
Assignment due date : Mar. 27th

Objective
I aim to apply Fourier Transform and Inverse Fourier Transform on image file and implement the Butter Worth Low Pass Filter.

Background
This is important because when doing image processing, we need to do several filtering. And it is easy to do the filtering in the frequency domain. That’s why we need to learn how to do FFT and IFFT.

I use the FFT and IFFT function that the TA gives us. Besides that, I just follow the assignments instructions and do the program.

Images and Descriptions

Fig.1(cake.pgm)

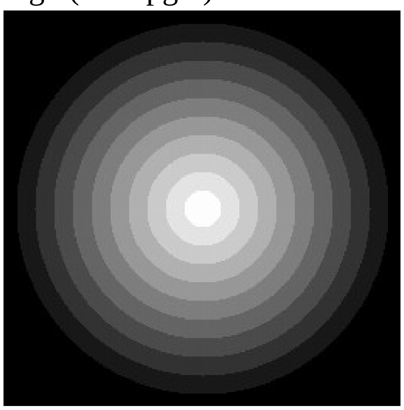



Fig.2(cake2.pgm)

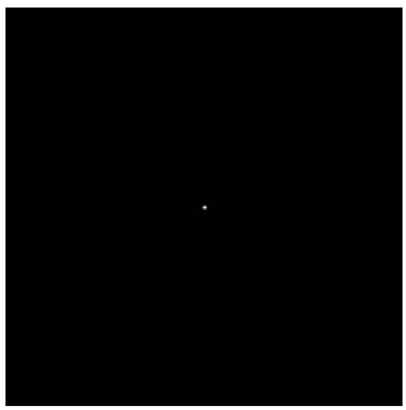



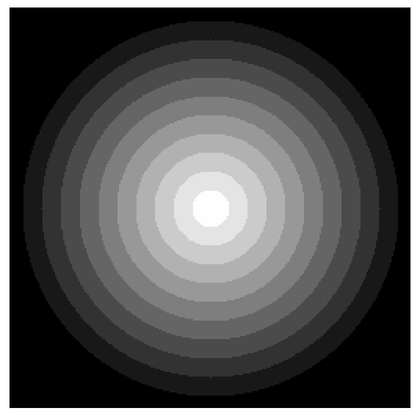
Fig.3(cake3.pgm)


Fig.1 is the original pgm image, then I implement the Fourier Transform and do the normalization to put in the center of the image , I get the Fig.2 image. Fig.3 is the image that after doing the Inverse Fourier Transform, which will look the same as the original one.

Fig.4 (mri.pgm)

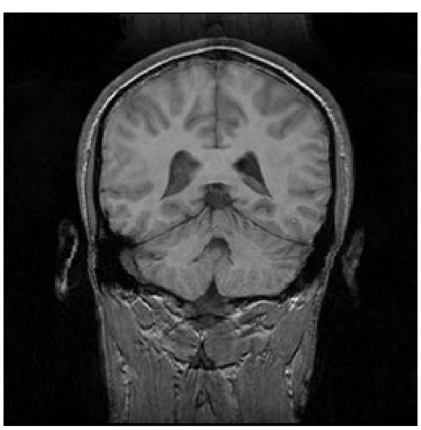



Fig.5(mri2.pgm)

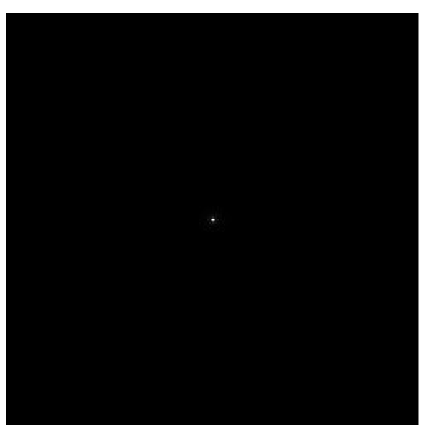



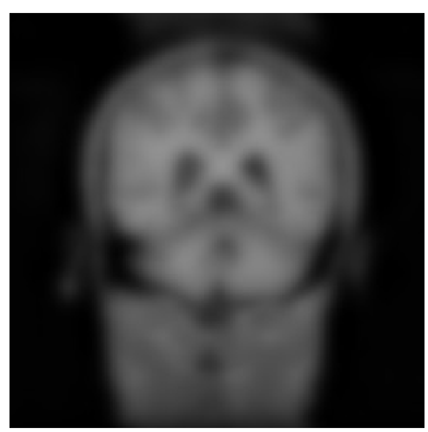
Fig.6(mri3.pgm)


Fig. 7(mri3-2.pgm)

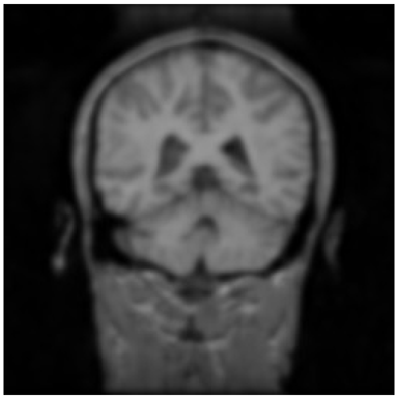


Fig.4 is the original pgm image, I implement the Fourier Transform and do the normalization to put in the center of the image and get the Fig.5 image, which is the spectrum of the image. Fig.6 is the image I implement Butter Worth Low Pass Filter and do the Inverse Fourier Transform, I set the order to 2 and D0 to 10; In Fig.7 ,I set the order to 2 and D0 to 25. We can see that the smaller the D0, the image would become much blur.

Conclusions

We learned from this assignment that we can use FFT and IFFT to do the image processing and be able to apply Butter Worth Low Pass Filter to make the image blurry.

Source Code

Part 1:

```
#include <stdio.h>
#include <stdlib.h>
#include "Four2.h"
#include "cplx.h"
#include <math.h>
#include <float.h>
#include <limits.h>

int main(int argc, char *argv[]){
        double min = DBL_MAX;     //set minimum doubled value
        double max = 0.;    //set maximum doubled value
        double *spectrum_m;       //spectrum magnitude

        unsigned i;
        unsigned n = 256;              //size of 2D data in both directions
        unsigned char *output_n;   //normalized spectrum output
        unsigned char *outputCake; //cake image output

        unsigned sizeX;     //image width
        unsigned sizeY;     //image height
        unsigned char *image;     //image 1D array
        unsigned levels;          //image brightness

        //read image from file
        FILE *iFile = fopen("cake.pgm","r");
        if(iFile == 0) return 1;
        if(3!=fscanf(iFile, "P5 %d %d %d ", &sizeX, &sizeY, &levels)) return 1;
```

```
image = (unsigned char *) malloc(sizeX*sizeY);
fread(image, sizeof(unsigned char), sizeX*sizeY, iFile);
fclose(iFile);

cplx *data;

data = (cplx*)calloc(sizeof(cplx),n*n);
spectrum_m = (double *)malloc(sizeof(double)*n*n);
output_n = (unsigned char *)malloc(n*n);
outputCake = (unsigned char *)malloc(n*n);

//put the image pixels values to data real part
if(data){
	for(int i=0; i<256; i++){
		for(int j=0; j<256; j++)
			data[i*256 + j].real = image[i*256 + j];
	}
}

//Fourier Transform
fft_Four2((float*) data,n,n,false);

//calculate the magnitude of spectrum and put it to spectrum_m
for(i = 0; i < n*n; i++){
spectrum_m[i] = sqrt(pow(data[i].real,2) + pow(data[i].imag,2));
}

//select the min and max in spectrum_m and do the normalization
for(i = 0; i < n*n; i++){
	if (spectrum_m[i] < min)
		min = spectrum_m[i];
	if (spectrum_m[i] > max)
		max = spectrum_m[i];
}
for(i = 0; i < n*n; i++){
	output_n[i] = 255*(spectrum_m[i]-min)/(max-min);
}

//write normalized spectrum magnitude image to file
iFile = fopen("cake2.pgm","w");
if(iFile == 0) return 1;
fprintf(iFile, "P5 %d %d %d ", sizeX, sizeY, 255);
fwrite(output_n, sizeof(unsigned char), sizeX*sizeY, iFile);
fclose(iFile);

//Inverse Fourier transform
fft_Four2((float*) data,n,n,true);

for(i = 0; i < n*n; i++){
	outputCake[i] = data[i].real;
}

//write output cake image to file
iFile = fopen("cake3.pgm","w");
if(iFile == 0) return 1;
fprintf(iFile, "P5 %d %d %d ", sizeX, sizeY, 255);
fwrite(outputCake, sizeof(unsigned char), sizeX*sizeY, iFile);
fclose(iFile);
```

```
	free(data);
	free(spectrum_m);
	free(output_n);
}
```

Part 2: (This code I set the D0 to 10 and output as mri3)

```
#include <stdio.h>
#include <stdlib.h>
#include "Four2.h"
#include "cplx.h"
#include <math.h>
#include <float.h>
#include <limits.h>

int main(int argc, char *argv[]){
	double min = DBL_MAX;	//set minimum doubled value
	double max = 0.;	//set maximum doubled value
	double *spectrum_m;	//spectrum magnitude

	unsigned i;
	unsigned n = 256;		//size of 2D data in both directions
	unsigned char *output_n;	//normalized spectrum output
	unsigned char *outputMRI;	//mri image output

	unsigned sizeX;	//image width
	unsigned sizeY;	//image height
	unsigned char *image;	//image 1D array
	unsigned levels;	//image brightness

	unsigned order = 2;	//LPF order
	float D;		//distance to the image center
	float H;
	unsigned D0 = 10;	//cut off value

	//read image from file
	FILE *iFile = fopen("mri.pgm","r");
	if(iFile == 0) return 1;
	if(3!=fscanf(iFile, "P5 %d %d %d ", &sizeX, &sizeY, &levels)) return 1;
	image = (unsigned char *) malloc(sizeX*sizeY);
	fread(image, sizeof(unsigned char), sizeX*sizeY, iFile);
	fclose(iFile);

	cplx *data;

	data = (cplx*)calloc(sizeof(cplx),n*n);
	spectrum_m = (double *)malloc(sizeof(double)*n*n);
	output_n = (unsigned char *)malloc(n*n);
	outputMRI = (unsigned char *)malloc(n*n);

	//put image pixel values to data real part
	if(data){
		for(int i=0; i<256; i++){
			for(int j=0; j<256; j++)
				data[i*256 + j].real = image[i*256 + j];
		}
	}
```

```
	//do Fourier transform
	fft_Four2((float*) data,n,n,false);

	//apply BWLPF to both data real part and imaginary part
	for(int i=0; i<256; i++)
		for(int j=0; j<256; j++){
			D = sqrt(pow((i - 128),2) + pow((j - 128),2));
			H = 1/(1 + pow((D/D0),2*order));
			data[i*256+j].real *= H;
		    data[i*256+j].imag *= H;
		}


	//calculate the magnitude of spectrum
	for(i = 0; i < n*n; i++){
	spectrum_m[i] = sqrt(pow(data[i].real,2) + pow(data[i].imag,2));
	}

	//select the min and max in spectrum_m and do the normalization
	for(i = 0; i < n*n; i++){
		if (spectrum_m[i] < min)
			min = spectrum_m[i];
		if (spectrum_m[i] > max)
			max = spectrum_m[i];
	}
	for(i = 0; i < n*n; i++){
		output_n[i] = 255*(spectrum_m[i]-min)/(max-min);
	}

	//write output mri image to file
	iFile = fopen("mri2.pgm","w");
	if(iFile == 0) return 1;
	fprintf(iFile, "P5 %d %d %d ", sizeX, sizeY, 255);
	fwrite(output_n, sizeof(unsigned char), sizeX*sizeY, iFile);
	fclose(iFile);

	//do Inverse Fourier transform
	fft_Four2((float*) data,n,n,true);

	for(i=0;i<n*n;i++)
		outputMRI[i] = data[i].real;

	//write mri image to file
	iFile = fopen("mri3.pgm","w");
	if(iFile == 0) return 1;
	fprintf(iFile, "P5 %d %d %d ", sizeX, sizeY, 255);
	fwrite(outputMRI, sizeof(unsigned char), sizeX*sizeY, iFile);
	fclose(iFile);

	free(data);
	free(spectrum_m);
	free(output_n);
}
```