University of Nevada, Reno

CS 474 — Image Processing

Assignment #4

Joshua Gleason

Instructor:
Dr. George Bebis

November 2, 2010

Contents

List of Figures

1 Introduction

2 Implementation

3 Results

4 Source Code

4.1 funcs.h

```
#ifndef JDG_FUNCTIONS
#define JDG_FUNCTIONS

#include "image.h"

namespace numrec
{

#define SWAP(a,b) tempr=(a);(a)=(b);(b)=tempr

void fft(float data[], unsigned long nn, int isign)
{
  unsigned long n,mmax,m,j,istep,i;
  double wtemp,wr,wpr,wpi,wi,theta;
  float tempr,tempi;

  n=nn << 1;
  j=1;
  for (i=1;i<n;i+=2) {
    if (j > i) {
      SWAP(data[j],data[i]);
      SWAP(data[j+1],data[i+1]);
    }
    m=n >> 1;
    while (m >= 2 && j > m) {
      j -= m;
      m >>= 1;
    }
    j += m;
  }
  mmax=2;
  while (n > mmax) {
    istep=mmax << 1;
    theta=isign*(6.28318530717959/mmax);
    wtemp=sin(0.5*theta);
    wpr = -2.0*wtemp*wtemp;
    wpi=sin(theta);
    wr=1.0;
    wi=0.0;
    for (m=1;m<mmax;m+=2) {
      for (i=m;i<=n;i+=istep) {
        j=i+mmax;
        tempr=wr*data[j]-wi*data[j+1];
        tempi=wr*data[j+1]+wi*data[j];
        data[j]=data[i]-tempr;
        data[j+1]=data[i+1]-tempi;
        data[i] += tempr;
        data[i+1] += tempi;
      }
      wr=(wtemp=wr)*wpr-wi*wpi+wr;
      wi=wi*wpr+wtemp*wpi+wi;
    }
    mmax=istep;
  }
```

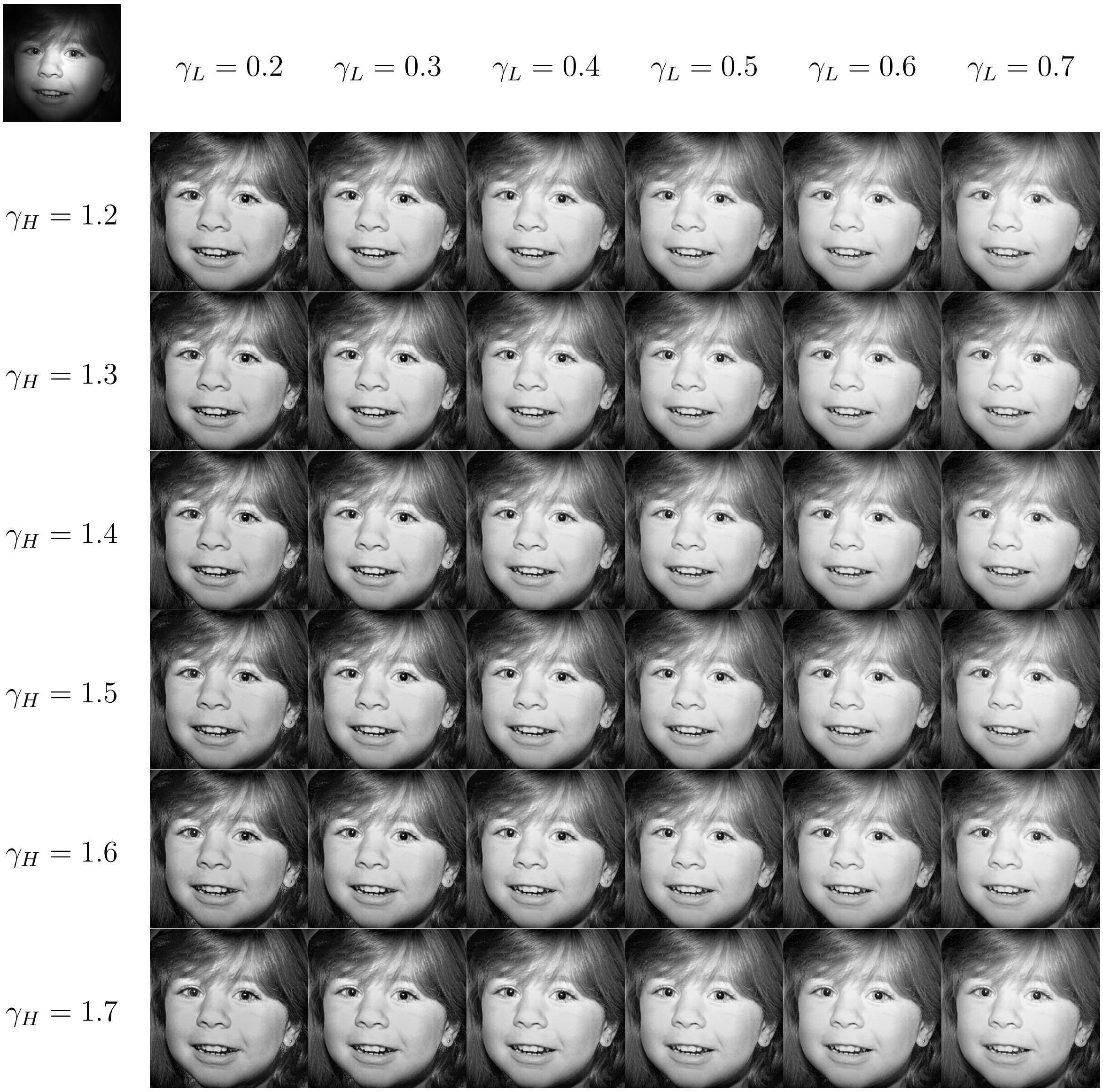
```
}
#undef SWAP

} // namespace numrec

namespace jdg
{

enum FilterType{ IDEAL=0, GAUSSIAN=1, BUTTERWORTH=2 };

template <class pType>
void buildLowPass( jdg::Image<pType>& filter, FilterType type, float
    cutoff1,
  float cutoff2=0.0, float cutoff3=0.0, bool freq_domain=false );

template <class pType>
void fft( Image<std::complex<pType> >& f, int val=1 );

template <class pType>
void convolve( Image<std::complex<pType> >& img,
  const Image<std::complex<pType> >& kernel, const PadWith=NEAREST );

template <class pType>
void fft( Image<std::complex<pType> >& f, int val )
{
  // resize to a power of 2
  int height = std::pow(2, std::ceil(log(f.getHeight())/log(2)));
  int width = std::pow(2, std::ceil(log(f.getWidth())/log(2)));

  // pad the image with zeros
  if ( height != f.getHeight() || width != f.getWidth() )
    f.pad( width, height, NEAREST );

  // large enough to hold rows or columns
  float* ary_vals = new float[std::max(width,height)*2];

  // perform 1D fft on all rows
  for ( int row = height-1; row >= 0; --row )
  {
    // build a row array
    for ( int i = width-1; i >= 0; --i )
    {
      // build the array for a row
      ary_vals[2*i] = static_cast<float>(f(i,row).real());
      ary_vals[2*i+1] = static_cast<float>(f(i,row).imag());

      // multiply by -1^(x+y)
      if ( (i+row)%2 != 0 && val >= 0 ) // odd
      {
        ary_vals[2*i] *= -1;
        ary_vals[2*i+1] *= -1;
      }
    }

    // find the fft of the row
    numrec::fft( ary_vals - 1, width, val );

    // put value back into image and multiply by 1/height
    for ( int i = width-1; i >= 0; --i )
    {
      f(i,row) = std::complex<pType>(
        static_cast<pType>(ary_vals[2*i]),     // real part
        static_cast<pType>(ary_vals[2*i+1]));  // imaginary part

      if ( val > 0 )
        f(i,row) *= 1.0/(height*width);
    }
  }

  // perform 1D fft on all columns
  for ( int col = width-1; col >= 0; --col )
  {
    for ( int i = height-1; i >= 0; --i )
    {
      ary_vals[2*i] = static_cast<float>(f(col,i).real());
      ary_vals[2*i+1] = static_cast<float>(f(col,i).imag());
    }

    numrec::fft( ary_vals - 1, height, val );

    for ( int i = height-1; i >= 0; --i )
      f(col,i) = std::complex<pType>(
        static_cast<pType>(ary_vals[2*i]),
        static_cast<pType>(ary_vals[2*i+1]));
  }

  delete [] ary_vals;
}

template <class pType>
void convolve( Image<std::complex<pType> >& img,
  const Image<std::complex<pType> >& kernel, const PadWith pad )
{
  Image<std::complex<pType> > kern = kernel;

  int origW = img.getWidth(), origH = img.getHeight();
  int dims =
    max( img.getWidth(), img.getHeight() ) +
    max( kern.getWidth(), kern.getHeight() );

  int shiftX = min(img.getWidth(), kernel.getWidth())/2;
  int shiftY = min(img.getHeight(), kernel.getHeight())/2;

  // pad images
  img.pad( dims, dims, pad, shiftX, shiftY );
  kern.pad( dims, dims );

  // fourier transform
  fft(img);
  fft(kern);

  // multiplication
  img *= kern;

  // invert fourier
  fft(img,-1);
  // unpad the image back to original size ZEROS because it’s efficient
  img.pad( origW, origH, jdg::ZEROS, -2*shiftX, -2*shiftY );
}

template <class pType>
void buildLowPass( jdg::Image<pType>& filter, FilterType type, float param1
    ,
  float param2, float param3, bool freq_domain )
{
  //filter.resizeCanvas(512,512);
  int width = filter.getWidth();
  int height = filter.getHeight();

  float startX = -(width-1) / 2.0,
      startY = -(height-1) / 2.0,
      stopX = -startX,
      stopY = -startY;

  //param1 = param1 * 0.5*sqrt(width*width+height*height);

  float param1_sqr = param1*param1;

  for ( float y = startY; y <= stopY; y+=1.0 )
    for ( float x = startX; x <= stopX; x+=1.0 )
      if ( type==IDEAL )
      {
        if ( sqrt(x*x+y*y) > param1 )
          filter(x-startX,y-startY) = 0;
        else
          filter(x-startX,y-startY) = 1;
      }
      else if ( type==GAUSSIAN )
      {
        filter(x-startX,y-startY) = exp(-(0.5*x*x+0.5*y*y)/(param1_sqr));
      }
      else if ( type==BUTTERWORTH )
      {
        if ( x != 0 && y != 0 )
          filter(x-startX,y-startY) = param2+param3/(1.0+param1*param1/((x*
              x+y*y)));
        else
          filter(x-startX,y-startY) = 0.0;
      }
  if ( !freq_domain )
    jdg::fft( filter, -1 );
}

}

#endif
```

4.2 main.cc

```
#include <iostream>
#include "funcs.h"
#include <sstream>
```

```
using namespace std;

complex<double> natlog( complex<double> val )
{
  return log(abs(val));
}

complex<double> exponential( complex<double> val )
{
  return exp(abs(val));
}

int main(int argc, char* argv[])
{
  for ( float YL = 0.2; YL <= 0.8; YL+=0.1 )
  for ( float YH = 1.2; YH <= 1.8; YH+=0.1 )
  {
    jdg::Image<complex<double> > a("images/girl.pgm");
    jdg::Image<complex<double> > filter(a.getWidth(),a.getHeight());
    jdg::Image<double> show;

    // step 1
    a.callFunc( &natlog );

    // step 2
    jdg::fft(a);

    // step 3
    // 0.3 and 1.3 ? seem to be good
    jdg::buildLowPass( filter, jdg::BUTTERWORTH, 1.8, YL, YH, true );

    a = a*filter;

    // step 4
    jdg::fft(a,-1);

    // step 5
    a.callFunc( &exponential );

    show = a;
    show.normalize( jdg::MINMAX_LOG, 0, 255 );
    //show.show();

    ostringstream sout;
    sout << "./images/girl_" << YL*10 << "_" << YH*10 << ".pgm";
    show.save(sout.str().c_str());
  }
  return 0;
}
```

5 Images


$\gamma_L = 0.2$
$\gamma_L = 0.3$
$\gamma_L = 0.4$
$\gamma_L = 0.5$
$\gamma_L = 0.6$
$\gamma_L = 0.7$
$\gamma_H = 1.2$
$\gamma_H = 1.3$
$\gamma_H = 1.4$
$\gamma_H = 1.5$
$\gamma_H = 1.6$
$\gamma_H = 1.7$