**PROJECT 1. Digital Image Filtering using Discrete Fourier Transform**

**Data Structure : Dynamically Allocated two-dimensional arrays**

**ESE 344 Software Techniques for Engineers**

**SUNY at Stony Brook, Murali Subbarao , Weight: 7%**

**DRAFT 2/7/2019 Subject to changes**

Digital images are often processed by computing the Discrete Fourier Transform (DFT). In some image filtering applications, the DFT of the image and the desired filters are computed, multiplied, and the inverse DFT of the result is computed. Transforms similar to DFT such as the Discrete Cosine Transform (DCT) are used in image/video compression. In this project, you will use two-dimensional arrays to store and compute the forward and inverse DFT of images. Memory for images will be determined at runtime. The separable property of the DFT along different dimensions will be used in computing the transform coefficients.

The input format is:

nr // number of rows.

nc // number of columns.

Image-data

Filter-data

**Example of input data:**

8

16

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7

3 3 4 5 6 8 9 0 1 1 2 3 4 5 6 7

4 3 4 5 6 8 9 0 1 1 2 3 4 5 6 7

5 3 4 5 6 8 9 0 1 1 2 3 4 5 6 7

6 3 4 5 6 8 9 0 1 1 2 3 4 5 6 7

7 3 4 5 6 8 9 0 1 1 2 3 4 5 6 7

8 3 4 5 6 8 9 0 1 1 2 3 4 5 6 7

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

2 0 0 0 0 0 9 0 1 2 3 4 5 6 7 1

3 0 0 0 0 0 9 0 1 2 3 4 5 6 7 2

4 3 4 5 6 8 9 0 1 1 2 3 4 5 6 7

5 3 4 5 6 8 1 1 3 4 5 6 7 8 5 3

6 0 0 0 0 0 9 0 1 2 3 4 5 6 7 4

7 3 4 5 6 8 9 0 1 2 3 4 5 6 7 5

8 0 0 0 0 0 9 0 1 2 3 4 5 6 7 6

**Steps:**

This will be explained in class.

* 1. Read input data—M , N, f, and h, ithat order. Let f denote the image, and h denote the filter. There will be no error in the put. You can use the simplest form of exception handling by using assert(condition) statements.
  2. See the example above for the input format. You can use the data above or your own made up data to test your program.
  3. Compute the DFT of the image f to get F and print it (complex numbers). Note that, input for f contains only the real numbers. If you want to multiply with complex matrices P and Q (see later), you can convert f to complex type by setting the imaginary part to be zeros. [F] = [P] \* [f]\*[Q]. See notes below.
  4. Compute the inverse DFT of F to get f’ and print it (print the real part). [f’]=[P’]\*[F]\*[Q’]. This output should be roughly the same as the input [f]. This verifies the DFT and IDFT.
  5. Compute the DFT of the filter h to get H and print it (complex numbers). The method is similar to that of computing [F] from [f]. Again, you may need to set the imaginary parts to be zeros in [h].
  6. Compute the inverse DFT of H to get h’ and print it (print the real part). Check that this is equal to h.
  7. Multiply **corresponding elements (not matrix multiplication)** of F and H (complex number multiplication) to get G. That is G[u][v]=H[u][v] F[u][v]. Note the range of u and v ( u= 0, 1,, 2, …, M-1, and v= 0, 1,, 2, …, N-1)
  8. Compute the inverse DFT of G to get g and print its real part. [g]=[P’]\*[G]\*[Q’].
  9. Compute the discrete circular convolution of f and h to get g’ and print the result.
  10. Compute the **sum of squared error** between the outputs g and g’ of steps (h) and (i). Due to the convolution property of DFT, g and g’ must be almost the same (except of round-off errors in the computations).

u,m= 0, 1, 2, …, M-1, and v,n= 0,1,2,3,…,N-1

Reference on complex numbers (STL): <http://www.cplusplus.com/reference/complex/>

Definition of DFT (Discrete Fourier Transform). This equation is for your information only. It is not needed in the program because this is implemented with matrix maultiplication. [F]=[P]\*[f]\*[Q].

Definition of IDFT (Inverse DFT). This equation is for your information only. It is not needed in the program because this is implemented with matrix maultiplication. [f’]=[P’]\*[F]\*[Q’].

**Computing DFT:**

Construct a complex valued MxM matrix [P]=P[u][m] = , and

Construct a complex valued NxN matrix [Q]=Q[v][n] =

DFT [F] of [f] is computed by [F]=[P]\*[f]\*[Q] ( matrix multiplication—it is associative).

**Computing inverse DFT (IDFT):**

Construct a complex valued MxM matrix [P’]=P’[u][m] = , and

Construct a complex valued NxN matrix [Q’]=Q’[v][n] =

Let the image be an MxN complex valued matrix—[f] = f[m][n], and let [F]=F[u][v] be its DFT which is a matrix of size MxN.

Then, in matrix notation, DFT is given by the matrix multiplications:

[F] = [P] \* [f]\* [Q].

And the inverse DFT is

[f’]=[P’] \* [F]\*[Q’].

Note that, it is not necessary to understand the Fourier transform to complete this project. You can just compute the matrices P, Q, P', Q', and perform matrix multiplication to get Fourier forward/inverse transform.

One thing you need to know is that,

exp(jx) = Cos(x)+j Sin(x). (j = sqrt(-1) = i).

That is, Cos(x) is the real part and Sin(x) is the imaginary part.

Therefore,

s = exp(j 2 π v n )

can be computed as, e.g.,

complex<double> r,s;

s= complex<double>( cos(2 π v n ) , sin (2 π v n ));

It is also possible to get s as:

r=complex<double>( 0.0 , 2 π v n );

s=exp(r);

**Circular Convolution:**

u,m= 0, 1, 2, …, M-1, and v,n= 0,1,2,3,…,N-1

An example of a program (modified version in the text book) for matrix multiplication is given below. You can change it to complete this project. This program also illustrates many other features (without comments or explanation) that you may need.

Computing the sum of squared difference error (SSD error): It is given by the difference of the real parts (ignore imaginary parts) of g an g’ as:

If everything is correct, then this error should be small.

// Matrix class and multiplication programs modified by Prof. M. Subbarao

#ifndef MATRIX\_H

#define MATRIX\_H

#include <iostream>

#include <vector>

#include <string>

#include <complex>

#include <math.h>

#include <cassert>

// Modified by Prof. Murali Subbarao, ESE 344, Jan. 2019

using namespace std;

template <typename Object>

class matrix

{

public:

matrix(int rows, int cols) : array(rows)

{

for (auto & thisRow : array)

thisRow.resize(cols);

}

matrix(vector<vector<Object>> v) : array{ v }

{ }

matrix(vector<vector<Object>> && v) : array{ std::move(v) }

{ }

const vector<Object> & operator[](int row) const

{

return array[row];

}

vector<Object> & operator[](int row)

{

return array[row];

}

int numrows() const

{

return array.size();

}

int numcols() const

{

return numrows() ? array[0].size() : 0;

}

void matprt()

{

int nr = numrows();

int nc = numcols();

for (int i = 0; i < nr; ++i) {

for (int j = 0; j < nc; ++j) {

cout << array[i][j] << " ";

}

cout << endl;

}

cout << endl <<endl;

}

private:

vector<vector<Object>> array;

};

#endif

/\*\*

\* Standard matrix multiplication.

\* Arrays start at 0.

\* Assumes a and b are square.

\*/

matrix<double> operator\*(const matrix<double> & a, const matrix<double> & b)

{

int nra = a.numrows();

int nca = a.numcols();

int nrb = b.numrows();

int ncb = b.numcols();

if (nca != nrb) {

cerr << "nca != nrb in matrix multiplication. Exiting." << endl;

cout << endl << "Enter any char to exit : " << endl;

char c;

cin >> c;

exit(0);

}

matrix<double> c{ nra, ncb };

for (int i = 0; i < nra; ++i) // Initialization

for (int j = 0; j < ncb; ++j)

c[i][j] = 0.0;

for (int i = 0; i < nra; ++i)

for (int j = 0; j < ncb; ++j)

for (int k = 0; k < nca; ++k)

c[i][j] += a[i][k] \* b[k][j];

return c;

}

/\*\*

\* Standard matrix multiplication.

\* Arrays start at 0.

\* Assumes a and b are square.

\*/

void matprint(const matrix<double> & a)

{

int nr = a.numrows();

int nc = a.numcols();

for (int i = 0; i < nr; ++i) {

for (int j = 0; j < nc; ++j) {

cout << a[i][j] << " ";

}

cout<< endl;

}

cout << endl<<endl;

}

int main()

{

int nr1, nc1, nr2, nc2;

char c;

cout << "Enter number of rows nr1 : (1 to 20) : " << endl;

cin >> nr1;

cout << "nr1 : " << nr1 << endl;

if ( (nr1<1) || (nr1>20) ) {

cout << "nr1 is out of range . " << endl;

assert((1 <= nr1) && (nr1 <= 20));

return 1;

// exit(1);

}

cout << "Enter number of columns nc1 : (1 to 20) : " << endl;

cin >> nc1;

cout << "nc1 : " << nc1 << endl;

if (!((1 <= nc1) && (nc1 <= 20))) {

cout << "nc1 is out of range . " << endl;

return 1;

}

cout << "Enter number of rows nr2 : (1 to 20) : " << endl;

cin >> nr2;

cout << "nr2 : " << nr2 << endl;

if (!((1 <= nr2) && (nr2 <= 20))) {

cout << "nr2 is out of range . " << endl;

return 1;

}

cout << "Enter number of columns nc2 : (1 to 20) : " << endl;

cin >> nc2;

cout << "nc2 : " << nc2 << endl;

if (!((1 <= nc2) && (nc2 <= 20))) {

cout << "nc1 is out of range . " << endl;

return 1;

}

matrix<double> mat1(nr1, nc1), mat2(nr2, nc2), mat3(nr1, nc2);

for (int i = 0; i < nr1; ++i) { // Initialization

for (int j = 0; j < nc1; ++j) {

mat1[i][j] = (double)(i + j+1);

}

}

for (int i = 0; i < nr2; ++i) { // Initialization

for (int j = 0; j < nc2; ++j) {

mat2[i][j] = (double)(i\*j+1);

}

}

mat3 = mat1 \* mat2;

cout <<endl<< "mat1 : " << endl;

matprint(mat1);

cout << endl << "mat2 : " << endl;

matprint(mat2);

cout << endl << "mat3 : " << endl;

matprint(mat3);

complex<double> p(1.0,2.0) , q(3.0 ,4.0),r;

cout << "p = " << p << "q = "<< q << endl;

p = complex<double>(5.0, 1.0);

cout << p.real() << " " << p.imag()<<endl;

r = p + q;

cout << "r = p+q = " << r<<endl;

r = p \* q;

cout << "r = p\*q = " << r << endl;

r = p /q;

cout << "r = p/q = " << r << endl;

matrix<complex <double>> mat4(nr1, nc1);

for (int i = 0; i < nr1; ++i) { // Initialization

for (int j = 0; j < nc1; ++j) {

mat4[i][j] = complex<double>((double)(i+1.0),(double)(j+1.0));

}

}

cout << endl << "mat4 : " << endl;

mat4.matprt();

cout << "exp(2.5) : " << exp(2.5) <<endl;

cout << "sqrt(5.0) : " << sqrt(5.0) <<endl;

cout << "exp(r) : " << exp(r) <<endl;

cout << "sin(2.5) : " << sin(2.5) << endl;

cout << "cos(2.5) : " << cos(2.5) << endl;

cout << endl << "Enter any char to continue : ";

cin >> c;

vector<vector<vector <double>>> a1;

vector<vector<vector <vector <vector <vector <vector <double>>>>>>> b1;

matrix<vector <double>> b2( 3 , 3 );

cout << endl << "Enter any char to continue : " << endl;

cin >> c;

return 0;

}