**Academic Integrity**

Every program requires a signed cover sheet:

|  |  |
| --- | --- |
|  | Program\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Programmer \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Academic Integrity Pledge  - All source code and documentation used in my program is either  my original work, or was derived, by me, from the source  code published in the textbook for this course or presented in class.  Any source code in this project that is derived from code published  elsewhere is documented as such so the original author receives due  credit.  - I have not used source code obtained from another student,  or any other unauthorized source, either modified or unmodified.  I have not helped another student write their program by providing  a printed or electronic copy of my solution.  - I have not discussed coding details about this project with anyone  other than my instructor. I understand that I may discuss the  concepts of this program with other students, and that another  student may help me debug my program. However, the responsibility  to write each program belongs solely to the program's author.  - I have violated neither the spirit nor letter of these restrictions.  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Date\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Signature of programmer |

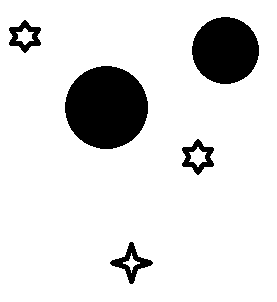
Nikolai Ruhe

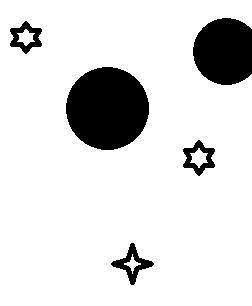
Algorithms Class

Dr. Duan

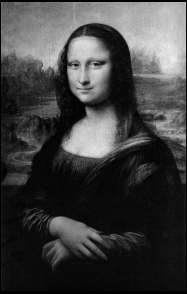
April 27, 2018

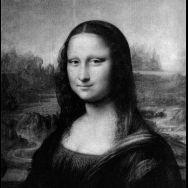
Vertical Seam Cut:





Horizontal Seam Cut:





Source Code:

#include <string>

#include <iostream>

#include <cstdlib>

#include <fstream>

#include <sstream>

using namespace std;

int\*\* uploadData(string FileName, int &xLength, int &yLength, int &max) {

int maxVal = 0;

string word = "", dimension = "", maxSize = "", pixelStr = "";

string xNum = "", yNum = "";

char letter;

bool p2Check = false, dimensionCheck = false, maxCheck = false, comment = false;

ifstream inFile(FileName);

getline(inFile, word);

while (word != "P2") {

getline(inFile, word);

}

getline(inFile, dimension);

while (dimension[0] == '#') {

getline(inFile, dimension);

}

getline(inFile, maxSize);

while (maxSize[0] == '#') {

getline(inFile, maxSize);

}

max = atoi(maxSize.c\_str());

bool splitD = false;

for (int i = 0; i < dimension.length(); i++) {

if (dimension[i] != ' ') {

if (splitD == false) {

xNum += dimension[i];

}

if (splitD == true) {

yNum += dimension[i];

}

}

else {

splitD = true;

}

}

xLength = atoi(xNum.c\_str());

yLength = atoi(yNum.c\_str());

int\*\* pixelMatrix = new int\*[yLength];

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

pixelMatrix[j] = new int[xLength];

}

for (int y = 0; y < yLength; y++) {

for (int x = 0; x < xLength; x++) {

inFile >> pixelMatrix[y][x];

}

}

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

for (int k = 0; k < xLength; k++) {

cout << pixelMatrix[j][k] << " ";

}

cout << endl;

}

return pixelMatrix;

}

int\*\* makeEMatrix(int\*\* pixelMatrix, int x, int y) {

int\*\* energy = new int\*[y];

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

energy[j] = new int[x];

}

for (int Yval = 0; Yval < y; Yval++) {

for (int Xval = 0; Xval < x; Xval++) {

if (Yval == 0 && Xval == 0) {

energy[Yval][Xval] = abs(pixelMatrix[Yval + 1][Xval] - pixelMatrix[Yval][Xval]) + abs(pixelMatrix[Yval][Xval + 1]- pixelMatrix[Yval][Xval]);

}

else if (Yval == 0 && Xval == x - 1) {

energy[Yval][Xval] = abs(pixelMatrix[Yval + 1][Xval]- pixelMatrix[Yval][Xval]) + abs(pixelMatrix[Yval][Xval - 1]- pixelMatrix[Yval][Xval]);

}

else if (Yval == y - 1 && Xval == 0) {

energy[Yval][Xval] = abs(pixelMatrix[Yval - 1][Xval]- pixelMatrix[Yval][Xval]) + abs(pixelMatrix[Yval][Xval + 1]- pixelMatrix[Yval][Xval]);

}

else if (Yval == y - 1 && Xval == x - 1) {

energy[Yval][Xval] = abs(pixelMatrix[Yval - 1][Xval]- pixelMatrix[Yval][Xval]) + abs(pixelMatrix[Yval][Xval - 1]- pixelMatrix[Yval][Xval]);

}

else if (Yval == 0){

energy[Yval][Xval] = abs(pixelMatrix[Yval][Xval - 1]- pixelMatrix[Yval][Xval]) + abs(pixelMatrix[Yval][Xval + 1]- pixelMatrix[Yval][Xval]) + abs(pixelMatrix[Yval + 1][Xval]- pixelMatrix[Yval][Xval]);

}

else if (Yval == y - 1) {

energy[Yval][Xval] = abs(pixelMatrix[Yval][Xval - 1]- pixelMatrix[Yval][Xval]) + abs(pixelMatrix[Yval][Xval + 1]- pixelMatrix[Yval][Xval]) + abs(pixelMatrix[Yval - 1][Xval]- pixelMatrix[Yval][Xval]);

}

else if (Xval == 0) {

energy[Yval][Xval] = abs(pixelMatrix[Yval - 1][Xval]- pixelMatrix[Yval][Xval]) + abs(pixelMatrix[Yval + 1][Xval]- pixelMatrix[Yval][Xval]) + abs(pixelMatrix[Yval][Xval + 1]- pixelMatrix[Yval][Xval]);

}

else if (Xval == x - 1) {

energy[Yval][Xval] = abs(pixelMatrix[Yval - 1][Xval]- pixelMatrix[Yval][Xval]) + abs(pixelMatrix[Yval + 1][Xval]- pixelMatrix[Yval][Xval]) + abs(pixelMatrix[Yval][Xval - 1]- pixelMatrix[Yval][Xval]);

}

else {

energy[Yval][Xval] = abs(pixelMatrix[Yval][Xval - 1] - pixelMatrix[Yval][Xval]) + abs(pixelMatrix[Yval][Xval + 1]- pixelMatrix[Yval][Xval]) + abs(pixelMatrix[Yval - 1][Xval]- pixelMatrix[Yval][Xval]) + abs(pixelMatrix[Yval + 1][Xval]- pixelMatrix[Yval][Xval]);

}

}

}

cout << "here is the energy matrix" << endl;

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

for (int k = 0; k < x; k++) {

cout << energy[j][k] << " ";

}

cout << endl;

}

return energy;

}

int\*\* carveHorizontal(int\*\* eMatrix, int\*\* &pixelMatrix, int x, int y) {

int Ylen = y - 1;

int Xlen = x;

int minIndex = 0, carve = 0, temp = 65000;

bool firstCol = true;

int\*\* newImg = new int\*[Ylen];

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

newImg[j] = new int[Xlen];

}

for (int X = 0; X < x; X++) {

for (int Y = 0; Y < y; Y++) {

if (firstCol == true) {

if (eMatrix[Y][X] < temp) {

temp = eMatrix[Y][X];

minIndex = Y;

carve = Y;

}

}

else {

if (Y <= carve + 1 && Y >= carve - 1) {

if (eMatrix[Y][X] < temp) {

temp = eMatrix[Y][X];

minIndex = Y;

}

}

}

}

firstCol = false;

temp = 65000;

eMatrix[minIndex][X] = -1;

carve = minIndex;

}

//cout << "here is the horizontal Seam" << endl;

//build the horizontal seam

int z = 0;

for (int a = 0; a < x; a++) {

for (int b = 0; b < y; b++) {

if (eMatrix[b][a] != -1) {

newImg[z][a] = pixelMatrix[b][a];

//cout << newImg[z][a] << " ";

z++;

}

}

z = 0;

//cout << endl;

}

return newImg;

}

int\*\* carveVertical(int\*\* eMatrix, int\*\* &pixelMatrix, int x, int y) {

int Ylen = y;

int Xlen = x-1;

int minIndex = 0, carve = 0, temp = 65000;

bool firstRow = true;

int\*\* newImg= new int\*[Ylen];

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

newImg[j] = new int[Xlen];

}

for (int Y = 0; Y < y; Y++) {

for (int X = 0; X < x; X++) {

if (firstRow == true) {

if (eMatrix[Y][X] < temp) {

temp = eMatrix[Y][X];

minIndex = X;

carve = X;

}

}

else {

if (X <= carve + 1 && X >= carve - 1) {

if (eMatrix[Y][X] < temp) {

temp = eMatrix[Y][X];

minIndex = X;

}

}

}

}

firstRow = false;

temp = 65000;

eMatrix[Y][minIndex] = -1;

carve = minIndex;

}

//cout << "here is the vertical Seam" << endl;

//build the vertical seam

int z = 0;

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

for (int k = 0; k < x; k++) {

if (eMatrix[j][k] != -1) {

newImg[j][z] = pixelMatrix[j][k];

//cout << newImg[j][z] << " ";

z++;

}

}

z = 0;

//cout << endl;

}

return newImg;

}

int main(int argc, char \*argv[]) {

int x = 0, y = 0, max = 0;

bool cutLong = false, cutWide = false;

int\*\* pixelMatrix = uploadData(argv[1], x, y, max);

int\*\* eMatrix = makeEMatrix(pixelMatrix, x, y);

int verticalSeams = stoi(argv[2]), horizontalSeams = stoi(argv[3]);

int\*\* verticalImgCarve = new int\*[y];

if (verticalSeams > 0) {

cutLong = true;

verticalImgCarve = carveVertical(eMatrix, pixelMatrix, x, y);

verticalSeams--;

x--;

}

while (verticalSeams > 0) {

verticalImgCarve = makeEMatrix(verticalImgCarve, x, y);

verticalImgCarve = carveVertical(verticalImgCarve, pixelMatrix, x, y);

verticalSeams--;

x--;

}

if (horizontalSeams > 0 && cutLong == false) {

verticalImgCarve = carveHorizontal(eMatrix, pixelMatrix, x, y);

horizontalSeams--;

y--;

}

else if (horizontalSeams > 0 && cutLong == true) {

verticalImgCarve = makeEMatrix(verticalImgCarve, x, y);

verticalImgCarve = carveHorizontal(verticalImgCarve, pixelMatrix, x, y);

horizontalSeams--;

y--;

}

while (horizontalSeams > 0) {

verticalImgCarve = makeEMatrix(verticalImgCarve, x, y);

verticalImgCarve = carveHorizontal(verticalImgCarve, pixelMatrix, x, y);

horizontalSeams--;

y--;

}

cout << "Here is the final Pic" << endl;

string fileType = "\_processed.pgm";

string newFileName = argv[1] + fileType;

ofstream myfile(newFileName);

if (myfile.is\_open())

{

myfile << "P2\n";

myfile << x;

myfile << " ";

myfile << y;

myfile << "\n";

myfile << max;

myfile << "\n";

for (int a = 0; a < y; a++) {

for (int b = 0; b < x; b++) {

cout << verticalImgCarve[a][b] << " ";

myfile << verticalImgCarve[a][b] << "\t";

}

myfile << "\n";

cout << endl;

}

myfile.close();

}

return 0;

}