Lab 4 Report – Basil Lin

In this lab, we implemented a region growing program together with the win32 API to create a graphical Windows program capable of growing a selected region of a specified file. Sample code for a Windows image editing GUI and the region growing algorithm were given to us. Using both of these, we were to create a graphical image editor capable of growing a region either continuously, or in a user trigger step mode.

Region growing works by checking a neighboring pixel with the average and variance of the surrounding pixels. Should the neighboring pixel pass the threshold for average and variance, the pixel will be allowed to join the region. In this program, the pixels joining the region are visualized by coloring in the original image with a user chosen color. For my specific project, I allowed the user to choose between red, green, and blue, for simplicities sake.

To start this program, I modified the .rc file of the Visual Studio project to include more menu options, including the ability to choose a color, the ability to choose step mode or play mode, and the ability to clear the image of the region growing edits to revert to the original loaded image. After modifying the menu, I wrote logic to make sure that different menu options would work together harmoniously, including checking and unchecking various menu options.

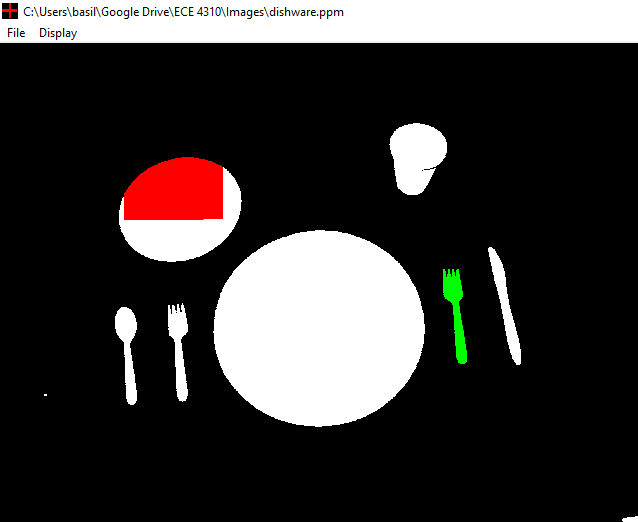
Once my menu worked correctly, I used the given region growing algorithm and combined it with the code for the GUI to enable the region growing options. I removed the loop for the code that goes over the entire image, as the user will click where they want the region growing to begin. In order to do this, I had to create multiple threads in order to handle switching between continuous region growing and region growing in step mode. In these threads, I used the given region growing algorithm together with the mouse value when a left click is registered to start the region growing process.

Should the user switch modes, the region growing may either be stepped by pressing or holding the letter ‘j’, or continuously grown at a rate of 1 pixel per millisecond. Switching modes can be done on the fly and does not require the region growing process to restart. Additionally, clicking on a different area of the picture will stop the current region growing and start region growing where the mouse was most recently left clicked. Again, mode switching between continuous and step modes is still available and will continue where the region is currently being grown.

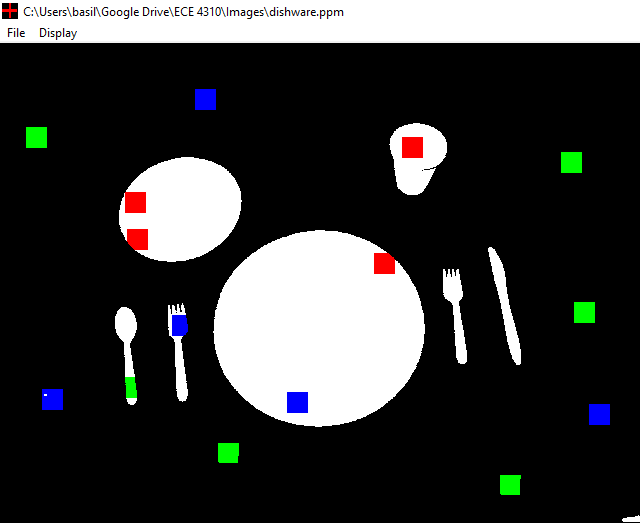
Next, I handled user input by creating a dialog box to change specified values within the region growing algorithm, such as the absolute difference in which a pixel is allowed to join, as well as the distance from the centroid of a region. With these changes, you can see how some pixels will stop being allowed to join the region as opposed to unmodified default thresholds.

Lastly, I implemented a few error checking measures to ensure that my program would work correctly. Originally, I had threads running multiple times, which caused a few issues that were fixed when I made the threads start in one place and made the threads stop when new threads were started. Additionally, I added a few extra lines of logic to ensure that a mode, a picture, and a color were chosen before the region growing algorithm could be run.

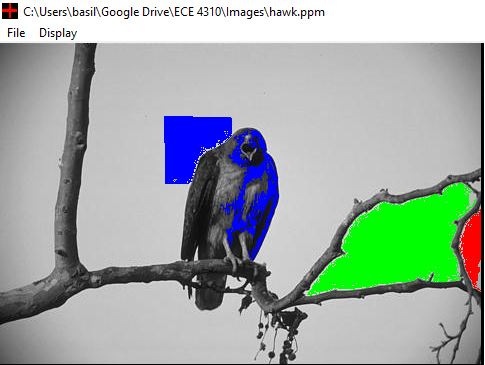
Here is an example of the region growing working. Notice that the red one was partially stopped to start the green one.



Here is another example of a limited range of 10 pixels from the centroid being allowed to join the region.



Lastly, another example of region growing working with a more complicated image.



Code:

#include <stdio.h>

#include <math.h>

#include <time.h>

#include <sys/timeb.h>

#include <windows.h>

#include <wingdi.h>

#include <winuser.h>

#include <process.h> /\* needed for multithreading \*/

#include "resource.h"

#include "globals.h"

#include "reg\_grow.h"

#include "resource1.h"

int APIENTRY WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,

LPTSTR lpCmdLine, int nCmdShow)

{

MSG msg;

HWND hWnd;

WNDCLASS wc;

wc.style = CS\_HREDRAW | CS\_VREDRAW;

wc.lpfnWndProc = (WNDPROC)WndProc;

wc.cbClsExtra = 0;

wc.cbWndExtra = 0;

wc.hInstance = hInstance;

wc.hIcon = LoadIcon(hInstance, "ID\_PLUS\_ICON");

wc.hCursor = LoadCursor(NULL, IDC\_ARROW);

wc.hbrBackground = (HBRUSH)(COLOR\_WINDOW + 1);

wc.lpszMenuName = "ID\_MAIN\_MENU";

wc.lpszClassName = "PLUS";

if (!RegisterClass(&wc))

return(FALSE);

hWnd = CreateWindow("PLUS", "plus program",

WS\_OVERLAPPEDWINDOW | WS\_HSCROLL | WS\_VSCROLL,

CW\_USEDEFAULT, 0, 400, 400, NULL, NULL, hInstance, NULL); //size of window

if (!hWnd)

return(FALSE);

ShowScrollBar(hWnd, SB\_BOTH, FALSE);

ShowWindow(hWnd, nCmdShow);

UpdateWindow(hWnd);

MainWnd = hWnd;

ShowPixelCoords = 0;

colorCheck = 0;

modeCheck = 0;

index = 0;

clicked = 0;

loaded = 0;

strcpy(filename, "");

OriginalImage = NULL;

labels = NULL;

indices = NULL;

ROWS = COLS = 0;

InvalidateRect(hWnd, NULL, TRUE);

UpdateWindow(hWnd);

while (GetMessage(&msg, NULL, 0, 0)) //event loop

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

return(msg.wParam);

}

LRESULT CALLBACK WndProc(HWND hWnd, UINT uMsg,

WPARAM wParam, LPARAM lParam)

{

HMENU hMenu;

OPENFILENAME ofn;

FILE \*fpt;

HDC hDC;

char header[320], text[320];

int BYTES, xPos, yPos;

int r, c, r2, c2;

int RegionSize, TotalRegions;

double avg, var;

void RegionGrow();

switch (uMsg)

{

case WM\_COMMAND:

switch (LOWORD(wParam))

{

case ID\_PIXELCOLOR\_RED:

colorCheck = 1;

break;

case ID\_PIXELCOLOR\_G:

colorCheck = 2;

break;

case ID\_PIXELCOLOR\_BLUE:

colorCheck = 3;

break;

case ID\_DISPLAY\_PLAYMODE:

modeCheck = 1;

break;

case ID\_DISPLAY\_STEPMODE:

modeCheck = 2;

break;

case ID\_DISPLAY\_ORIGINALIMAGE:

playThreadRunning = 0;

index = 0;

clicked = 0;

if (labels != NULL) {

free(labels);

labels = NULL;

}

if (indices != NULL) {

free(indices);

indices = NULL;

}

/\* segmentation image = labels; calloc initializes all labels to 0 \*/

labels = (unsigned char \*)calloc(ROWS\*COLS, sizeof(unsigned char));

/\* used to quickly erase small grown regions \*/

indices = (int \*)calloc(ROWS\*COLS, sizeof(int));

PaintImage();

break;

case ID\_DISPLAY\_CHANGEPIXELJOINING:

DialogBox(GetModuleHandle(NULL),

MAKEINTRESOURCE(IDD\_DIALOG1),

hWnd,

dlgProc);

ShowWindow(dlgProc, SW\_SHOW);

break;

case ID\_SHOWPIXELCOORDS:

ShowPixelCoords = (ShowPixelCoords + 1) % 2;

PaintImage();

break;

case ID\_FILE\_LOAD:

index = 0;

clicked = 0;

playThreadRunning = 0;

if (OriginalImage != NULL)

{

free(OriginalImage);

OriginalImage = NULL;

}

if (labels != NULL) {

free(labels);

labels = NULL;

}

if (indices != NULL) {

free(indices);

indices = NULL;

}

memset(&(ofn), 0, sizeof(ofn));

ofn.lStructSize = sizeof(ofn);

ofn.lpstrFile = filename;

filename[0] = 0;

ofn.nMaxFile = MAX\_FILENAME\_CHARS;

ofn.Flags = OFN\_EXPLORER | OFN\_HIDEREADONLY;

ofn.lpstrFilter = "PPM files\0\*.ppm\0All files\0\*.\*\0\0";

if (!(GetOpenFileName(&ofn)) || filename[0] == '\0')

break; /\* user cancelled load \*/

if ((fpt = fopen(filename, "rb")) == NULL)

{

MessageBox(NULL, "Unable to open file", filename, MB\_OK | MB\_APPLMODAL);

break;

}

fscanf(fpt, "%s %d %d %d", header, &COLS, &ROWS, &BYTES);

if (strcmp(header, "P5") != 0 || BYTES != 255)

{

MessageBox(NULL, "Not a PPM (P5 greyscale) image", filename, MB\_OK | MB\_APPLMODAL);

fclose(fpt);

break;

}

OriginalImage = (unsigned char \*)calloc(ROWS\*COLS, 1);

/\* segmentation image = labels; calloc initializes all labels to 0 \*/

labels = (unsigned char \*)calloc(ROWS\*COLS, sizeof(unsigned char));

/\* used to quickly erase small grown regions \*/

indices = (int \*)calloc(ROWS\*COLS, sizeof(int));

header[0] = fgetc(fpt); /\* whitespace character after header \*/

fread(OriginalImage, 1, ROWS\*COLS, fpt);

fclose(fpt);

SetWindowText(hWnd, filename);

PaintImage();

loaded = 1;

break;

case ID\_FILE\_QUIT:

DestroyWindow(hWnd);

break;

}

break;

case WM\_SIZE: /\* could be used to detect when window size changes \*/

PaintImage();

return(DefWindowProc(hWnd, uMsg, wParam, lParam));

break;

case WM\_PAINT:

PaintImage();

return(DefWindowProc(hWnd, uMsg, wParam, lParam));

break;

case WM\_LBUTTONDOWN: //left button click

if (loaded == 1 && colorF != 0 && modeF != 0) {

playThreadRunning = 0;

if (clicked == 1) {

index = 0;

if (labels != NULL) {

free(labels);

labels = NULL;

}

if (indices != NULL) {

free(indices);

indices = NULL;

}

labels = (unsigned char \*)calloc(ROWS\*COLS, sizeof(unsigned char));

/\* used to quickly erase small grown regions \*/

indices = (int \*)calloc(ROWS\*COLS, sizeof(int));

}

stepy = HIWORD(lParam);

stepx = LOWORD(lParam);

clicked = 1;

}

break;

case WM\_MOUSEMOVE: //moving mouse

if (ShowPixelCoords == 1)

{

xPos = LOWORD(lParam);

yPos = HIWORD(lParam);

if (xPos >= 0 && xPos < COLS && yPos >= 0 && yPos < ROWS)

{

sprintf(text, "%d,%d=>%d ", xPos, yPos, OriginalImage[yPos\*COLS + xPos]);

hDC = GetDC(MainWnd);

TextOut(hDC, 0, 0, text, strlen(text)); /\* draw text on the window \*/ //also a printf

ReleaseDC(MainWnd, hDC);

}

}

return(DefWindowProc(hWnd, uMsg, wParam, lParam));

break;

case WM\_KEYDOWN: //any key

if ((wParam == 'j' || wParam == 'J') && modeF == 2 && clicked == 1) {

r = stepy;

c = stepx;

TotalRegions = 0;

avg = var = 0.0; /\* compute average and variance in 7x7 window \*/

for (r2 = -3; r2 <= 3; r2++)

for (c2 = -3; c2 <= 3; c2++)

avg += (double)(OriginalImage[(r + r2)\*COLS + (c + c2)]);

avg /= 49.0;

for (r2 = -3; r2 <= 3; r2++)

for (c2 = -3; c2 <= 3; c2++)

var += SQR(avg - (double)OriginalImage[(r + r2)\*COLS + (c + c2)]);

var = sqrt(var) / 49.0;

if (var < 1.0) {

TotalRegions++; /\* condition for seeding a new region is low var \*/

RegionGrow(OriginalImage, labels, ROWS, COLS, r, c, 0, TotalRegions, indices, &RegionSize);

playThreadRunning = 0;

\_beginthread(stepThread, 0, MainWnd); /\* start up a child thread to do other work while this thread continues GUI \*/

}

for (int i = 0; i < ROWS\*COLS; i++) {

if (indices[i] != 0) {

if (indices[i] / COLS < r - range || indices[i] / COLS > r + range || indices[i] % COLS < c - range || indices[i] % COLS > c + range || OriginalImage[indices[i]] > avg + difference) {

indices[i] = 0;

}

}

}

}

if (wParam == 's' || wParam == 'S')

PostMessage(MainWnd, WM\_COMMAND, ID\_SHOWPIXELCOORDS, 0); /\* send message to self \*/

return(DefWindowProc(hWnd, uMsg, wParam, lParam));

break;

case WM\_HSCROLL: /\* this event could be used to change what part of the image to draw \*/

PaintImage(); /\* direct PaintImage calls eliminate flicker; the alternative is InvalidateRect(hWnd,NULL,TRUE); UpdateWindow(hWnd); \*/

return(DefWindowProc(hWnd, uMsg, wParam, lParam));

break;

case WM\_VSCROLL: /\* this event could be used to change what part of the image to draw \*/

PaintImage();

return(DefWindowProc(hWnd, uMsg, wParam, lParam));

break;

case WM\_DESTROY:

PostQuitMessage(0);

break;

default:

return(DefWindowProc(hWnd, uMsg, wParam, lParam));

break;

}

hMenu = GetMenu(MainWnd);

if (ShowPixelCoords == 1)

CheckMenuItem(hMenu, ID\_SHOWPIXELCOORDS, MF\_CHECKED); /\* you can also call EnableMenuItem() to grey(disable) an option \*/

else

CheckMenuItem(hMenu, ID\_SHOWPIXELCOORDS, MF\_UNCHECKED);

/\* get color \*/

if (colorCheck == 1) {

colorF = 1;

CheckMenuItem(hMenu, ID\_PIXELCOLOR\_RED, MF\_CHECKED);

CheckMenuItem(hMenu, ID\_PIXELCOLOR\_G, MF\_UNCHECKED);

CheckMenuItem(hMenu, ID\_PIXELCOLOR\_BLUE, MF\_UNCHECKED);

}

else if (colorCheck == 2) {

colorF = 2;

CheckMenuItem(hMenu, ID\_PIXELCOLOR\_RED, MF\_UNCHECKED);

CheckMenuItem(hMenu, ID\_PIXELCOLOR\_G, MF\_CHECKED);

CheckMenuItem(hMenu, ID\_PIXELCOLOR\_BLUE, MF\_UNCHECKED);

}

else if (colorCheck == 3) {

colorF = 3;

CheckMenuItem(hMenu, ID\_PIXELCOLOR\_RED, MF\_UNCHECKED);

CheckMenuItem(hMenu, ID\_PIXELCOLOR\_G, MF\_UNCHECKED);

CheckMenuItem(hMenu, ID\_PIXELCOLOR\_BLUE, MF\_CHECKED);

}

else {

colorF = 0;

CheckMenuItem(hMenu, ID\_PIXELCOLOR\_RED, MF\_UNCHECKED);

CheckMenuItem(hMenu, ID\_PIXELCOLOR\_G, MF\_UNCHECKED);

CheckMenuItem(hMenu, ID\_PIXELCOLOR\_BLUE, MF\_UNCHECKED);

}

/\* get mode \*/

if (modeCheck == 1 && playThreadRunning == 0) {

modeF = 1;

playThreadRunning = 0;

CheckMenuItem(hMenu, ID\_DISPLAY\_PLAYMODE, MF\_CHECKED);

CheckMenuItem(hMenu, ID\_DISPLAY\_STEPMODE, MF\_UNCHECKED);

if (clicked == 1) {

r = stepy;

c = stepx;

TotalRegions = 0;

avg = var = 0.0; /\* compute average and variance in 7x7 window \*/

for (r2 = -3; r2 <= 3; r2++)

for (c2 = -3; c2 <= 3; c2++)

avg += (double)(OriginalImage[(r + r2)\*COLS + (c + c2)]);

avg /= 49.0;

for (r2 = -3; r2 <= 3; r2++)

for (c2 = -3; c2 <= 3; c2++)

var += SQR(avg - (double)OriginalImage[(r + r2)\*COLS + (c + c2)]);

var = sqrt(var) / 49.0;

if (var < 1.0) {

TotalRegions++; /\* condition for seeding a new region is low var \*/

RegionGrow(OriginalImage, labels, ROWS, COLS, r, c, 0, TotalRegions, indices, &RegionSize);

playThreadRunning = 1;

\_beginthread(playThread, 0, MainWnd); /\* start up a child thread to do other work while this thread continues GUI \*/

}

for (int i = 0; i < ROWS\*COLS; i++) {

if (indices[i] != 0) {

if (indices[i] / COLS < r - range || indices[i] / COLS > r + range || indices[i] % COLS < c - range || indices[i] % COLS > c + range || OriginalImage[indices[i]] > avg + difference) {

indices[i] = 0;

}

}

}

}

}

else if (modeCheck == 2) {

modeF = 2;

playThreadRunning = 0;

CheckMenuItem(hMenu, ID\_DISPLAY\_PLAYMODE, MF\_UNCHECKED);

CheckMenuItem(hMenu, ID\_DISPLAY\_STEPMODE, MF\_CHECKED);

if (clicked == 1) {

r = stepy;

c = stepx;

TotalRegions = 0;

avg = var = 0.0; /\* compute average and variance in 7x7 window \*/

for (r2 = -3; r2 <= 3; r2++)

for (c2 = -3; c2 <= 3; c2++)

avg += (double)(OriginalImage[(r + r2)\*COLS + (c + c2)]);

avg /= 49.0;

for (r2 = -3; r2 <= 3; r2++)

for (c2 = -3; c2 <= 3; c2++)

var += SQR(avg - (double)OriginalImage[(r + r2)\*COLS + (c + c2)]);

var = sqrt(var) / 49.0;

if (var < 1.0) {

TotalRegions++; /\* condition for seeding a new region is low var \*/

RegionGrow(OriginalImage, labels, ROWS, COLS, r, c, 0, TotalRegions, indices, &RegionSize);

\_beginthread(stepThread, 0, MainWnd); /\* start up a child thread to do other work while this thread continues GUI \*/

}

}

}

DrawMenuBar(hWnd);

return(0L);

}

void PaintImage()

{

PAINTSTRUCT Painter;

HDC hDC;

BITMAPINFOHEADER bm\_info\_header;

BITMAPINFO \*bm\_info;

int i, r, c, DISPLAY\_ROWS, DISPLAY\_COLS;

unsigned char \*DisplayImage;

if (OriginalImage == NULL)

return; /\* no image to draw \*/

/\* Windows pads to 4-byte boundaries. We have to round the size up to 4 in each dimension, filling with black. \*/

DISPLAY\_ROWS = ROWS;

DISPLAY\_COLS = COLS;

if (DISPLAY\_ROWS % 4 != 0)

DISPLAY\_ROWS = (DISPLAY\_ROWS / 4 + 1) \* 4;

if (DISPLAY\_COLS % 4 != 0)

DISPLAY\_COLS = (DISPLAY\_COLS / 4 + 1) \* 4;

DisplayImage = (unsigned char \*)calloc(DISPLAY\_ROWS\*DISPLAY\_COLS, 1);

for (r = 0; r < ROWS; r++)

for (c = 0; c < COLS; c++)

DisplayImage[r\*DISPLAY\_COLS + c] = OriginalImage[r\*COLS + c];

BeginPaint(MainWnd, &Painter);

hDC = GetDC(MainWnd);

bm\_info\_header.biSize = sizeof(BITMAPINFOHEADER);

bm\_info\_header.biWidth = DISPLAY\_COLS;

bm\_info\_header.biHeight = -DISPLAY\_ROWS;

bm\_info\_header.biPlanes = 1;

bm\_info\_header.biBitCount = 8;

bm\_info\_header.biCompression = BI\_RGB;

bm\_info\_header.biSizeImage = 0;

bm\_info\_header.biXPelsPerMeter = 0;

bm\_info\_header.biYPelsPerMeter = 0;

bm\_info\_header.biClrUsed = 256;

bm\_info\_header.biClrImportant = 256;

bm\_info = (BITMAPINFO \*)calloc(1, sizeof(BITMAPINFO) + 256 \* sizeof(RGBQUAD));

bm\_info->bmiHeader = bm\_info\_header;

for (i = 0; i < 256; i++)

{

bm\_info->bmiColors[i].rgbBlue = bm\_info->bmiColors[i].rgbGreen = bm\_info->bmiColors[i].rgbRed = i;

bm\_info->bmiColors[i].rgbReserved = 0;

}

SetDIBitsToDevice(hDC, 0, 0, DISPLAY\_COLS, DISPLAY\_ROWS, 0, 0,

0, /\* first scan line \*/

DISPLAY\_ROWS, /\* number of scan lines \*/

DisplayImage, bm\_info, DIB\_RGB\_COLORS);

ReleaseDC(MainWnd, hDC);

EndPaint(MainWnd, &Painter);

free(DisplayImage);

free(bm\_info);

}

void playThread(HWND AnimationWindowHandle) {

HDC hDC;

while (playThreadRunning == 1 && index < ROWS\*COLS) {

hDC = GetDC(MainWnd);

if (indices[index] != 0) {

if (colorF == 1) {

SetPixel(hDC, indices[index] % COLS, indices[index] / COLS, RGB(255, 0, 0)); /\* color the animation pixel red \*/

}

else if (colorF == 2) {

SetPixel(hDC, indices[index] % COLS, indices[index] / COLS, RGB(0, 255, 0)); /\* color the animation pixel green \*/

}

else if (colorF == 3) {

SetPixel(hDC, indices[index] % COLS, indices[index] / COLS, RGB(0, 0, 255)); /\* color the animation pixel blue \*/

}

}

index++;

ReleaseDC(MainWnd, hDC);

Sleep(1); /\* pause 1 ms \*/

}

playThreadRunning = 0;

}

void stepThread(HWND AnimationWindowHandle) {

HDC hDC;

if (index >= ROWS \* COLS) index = 0;

hDC = GetDC(MainWnd);

if (indices[index] != 0) {

if (colorF == 1) {

SetPixel(hDC, indices[index] % COLS, indices[index] / COLS, RGB(255, 0, 0)); /\* color the animation pixel red \*/

}

else if (colorF == 2) {

SetPixel(hDC, indices[index] % COLS, indices[index] / COLS, RGB(0, 255, 0)); /\* color the animation pixel green \*/

}

else if (colorF == 3) {

SetPixel(hDC, indices[index] % COLS, indices[index] / COLS, RGB(0, 0, 255)); /\* color the animation pixel blue \*/

}

}

index++;

ReleaseDC(MainWnd, hDC);

}

INT\_PTR CALLBACK dlgProc(HWND hDlg, UINT Msg,

WPARAM wParam, LPARAM lParam)

{

char buffer1[256];

char buffer2[256];

switch (Msg) {

case WM\_INITDIALOG:

return TRUE;

case WM\_COMMAND:

switch (LOWORD(wParam)) {

case IDOK:

GetDlgItemText(hDlg, IDC\_EDIT1, buffer1, 256);

range = atoi(buffer1);

GetDlgItemText(hDlg, IDC\_EDIT2, buffer2, 256);

difference = atoi(buffer2);

EndDialog(hDlg, wParam);

break;

case IDCANCEL:

range = 99999;

difference = 255;

EndDialog(hDlg, IDCANCEL);

break;

}

break;

default:

return FALSE;

}

return TRUE;

}

/\* Given an image, a starting point, and a label, this routine

\*\* paint-fills (8-connected) the area with the given new label

\*\* according to the given criteria (pixels close to the average

\*\* intensity of the growing region are allowed to join).

\*/

#define MAX\_QUEUE 10000 /\* max perimeter size (pixels) of border wavefront \*/

void RegionGrow(unsigned char \*image, /\* image data \*/

unsigned char \*labels, /\* segmentation labels \*/

int ROWS, int COLS, /\* size of image \*/

int r, int c, /\* pixel to paint from \*/

int paint\_over\_label, /\* image label to paint over \*/

int new\_label, /\* image label for painting \*/

int \*indices, /\* output: indices of pixels painted \*/

int \*count) /\* output: count of pixels painted \*/

{

int r2, c2;

int queue[MAX\_QUEUE], qh, qt;

int average, total; /\* average and total intensity in growing region \*/

\*count = 0;

if (labels[r\*COLS + c] != paint\_over\_label)

return;

labels[r\*COLS + c] = new\_label;

average = total = (int)image[r\*COLS + c];

if (indices != NULL)

indices[0] = r \* COLS + c;

queue[0] = r \* COLS + c;

qh = 1; /\* queue head \*/

qt = 0; /\* queue tail \*/

(\*count) = 1;

while (qt != qh)

{

if ((\*count) % 50 == 0) /\* recalculate average after each 50 pixels join \*/

{

average = total / (\*count);

// printf("new avg=%d\n",average);

}

for (r2 = -1; r2 <= 1; r2++)

for (c2 = -1; c2 <= 1; c2++)

{

if (r2 == 0 && c2 == 0)

continue;

if ((queue[qt] / COLS + r2) < 0 || (queue[qt] / COLS + r2) >= ROWS ||

(queue[qt] % COLS + c2) < 0 || (queue[qt] % COLS + c2) >= COLS)

continue;

if (labels[(queue[qt] / COLS + r2)\*COLS + queue[qt] % COLS + c2] != paint\_over\_label)

continue;

/\* test criteria to join region \*/

if (abs((int)(image[(queue[qt] / COLS + r2)\*COLS + queue[qt] % COLS + c2])

- average) > 10)

continue;

labels[(queue[qt] / COLS + r2)\*COLS + queue[qt] % COLS + c2] = new\_label;

if (indices != NULL)

indices[\*count] = (queue[qt] / COLS + r2)\*COLS + queue[qt] % COLS + c2;

total += image[(queue[qt] / COLS + r2)\*COLS + queue[qt] % COLS + c2];

(\*count)++;

queue[qh] = (queue[qt] / COLS + r2)\*COLS + queue[qt] % COLS + c2;

qh = (qh + 1) % MAX\_QUEUE;

if (qh == qt)

{

printf("Max queue size exceeded\n");

exit(0);

}

}

qt = (qt + 1) % MAX\_QUEUE;

}

}