ECE 6930 Introduction to Computer Vision

Semester Project

Food Segmentation

Sharan Rajendran

1 Introduction

Active Contours or snakes are an important tool in Computer Vision. They are used to identify and segment objects in an image. This contours or snakes can be used to detect necessary objects in an image, which can be used to track the segmented objects and such. This project implements an active contour algorithm in a GUI format. This allows the user to semi-automatically segment the images by choosing the initial contour points.

2 Methods

2.1 Initial Contour Points

After the image is loaded, the program has to obtain the initial contour points as input from the user. This is done in two ways. Clicking the left mouse button and dragging it around the object draws a red line along the path of the mouse pointer. This red line is draw by coloring a 5x5 square centered around the current position of the mouse. Another method of choosing contour points is to click the right button at a point inside the object, which results in a set of contour points placed in a circle around the mouse position.

During the process of selecting the contour points, two contours are activated depending on the left or right click of the mouse. The left button option stores the selected contour points to be used for a Rubber Band Model and the right click option used the points for a Balloon model. For the Rubber Band model, the selected points are downsampled by choosing every fifth point in the contour line drawn. For the Balloon model, every third point is selected.

2.2 Activation of Contour

After the contour has been drawn , each respective model has its own start key. In order to start the Rubber Band model, key 'R' has to be pressed while for the Balloon model key 'B' is to be pressed. These keys activate the algorithm to be used for moving the contour or snake. After completion of the contour, key 'A' is to be pressed to indicate that no changes will be made to the final contour. The algorithm used for each contour model is explained below.

For both the contour models, three external energy terms and two internal energy terms have been used. The external energies used are the Sobel Gradient image of the input image,image intensity and the edges obtained by comparing the color intensity of the input RGB image. The Sobel image is squared before being used as an external energy term. Thus, the external energy of the image is given by,

$$E_{External} = -w_{grad} * E_{grad} + w_{Intensity} * E_{Intensity} + w_{Coloredges} * E_{EdgesBasedonColor} \tag{1}$$

where, $\mathrm{E_{grad}}$ is the square of the Sobel gradient, $\mathrm{E_{Intensity}}$ contains the intensity values of the image and $\mathrm{E_{Edges\ Based\ on\ Color}}$ contains the edges obtained by comparing the RGB values of the four neighbors. The weiights, $\mathrm{w_{grad}}$, $\mathrm{w_{Intensity}}$ and $\mathrm{w_{Color\_edges}}$ have the same absolute value for both models. This external energy looks to pull the contours toward the most of the boundary outlines of the food in the input image.

The internal energy consists of two energy term, curvature and elasticity. Given a set of points $\mathrm{P_i}$ where i=1,.....,n, with n being the total number of points in the contour, then the elasticity and the curvature is given by,

$$Elasticity = ||P_{i+1} - P_i||^2 \tag{2}$$

$$Curvature = ||P_{i+1} - 2P_i + P_{i-1}||^2 \tag{3}$$

For the Rubber band model, the weights for the internal enrgy are chosen such that it causes the active contour to shrink, while for the Balloon model, the weights shoudl cause the contour to expand. The internal and external enrgy terms are calculated in a 7x7 window centered around the contour point. It then looks to find the pixel positon at which the total energy, which is the sum of all the energy terms, is minimum. During each iteration, every contour point is moved to a position at which the energy is minimum.

2.3 Moving Contour Points Manually

After the contour is done moving, the user can choose to move contour points that are not at the required final position. Those when the user clicks and holds down at the contour point or anywhere in the 5x5 window centered around it, the user can opt to move the contour. This activates another contour algorithm, which only looks to move the closest ten contour points. Thus, if the user is done using the contour, pressing the key 'A' allows the program to finalise the contour points, while giving the user to generate a new set of contour points on the image.

3 Results

The performance of the contour points for the Rubber Band and Balloon model were acceptable, whereas the option of moving the contour points failed to do really well. Moving a single contour point was achieved but it did not result in any huge changes in the preexisting contour. The results of the active conotour models with the test images are given below.

The results of the other test results are included in the appendix.

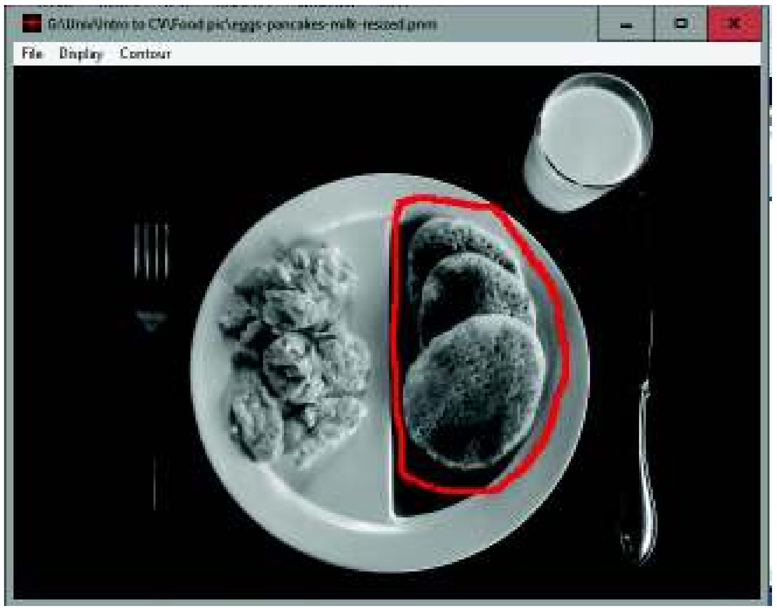

Figure 1: Rubber Band Model - Start Position

3.1 Suggested Improvements

Improvements that can be included to the code include a faster algorithm for computation the contours next position. Threading was implemented for the rubber band and balloon models, but making use of the threads while keeping the contours active for point movement was attempted and failed. So, the final program did not implement threading. Thus, a method of moving points within a thread could be implemented by making use of mouse inputs.

Another problem encountered was the fixed number of points. For example, in the Balloon model, since the points are downsampled and the initial contour has a radius of 10 pixels, the number of points (14) was not enough when the balloon model was used to detect a large object. For the Rubber Band model, multiple points converge together. Thus, an algorithm could be implemented such that the points can be deleted or added depending on necessity. The option of adding points can also be extended to contour point movement as explained in the following paragraph.

The option of moving contour points implemented allows the user to only use existing contour points. The option of allowing the user to choose any point along the contour (not necessarily an active contour point) can help improve the final performance. An attempt was made to draw lines between the contour points, so that the user can know the region form which he can move the contour. An algorithm, called Bresenham's Line Algorithm allows the computation of points which are to be colored between the start and end points.

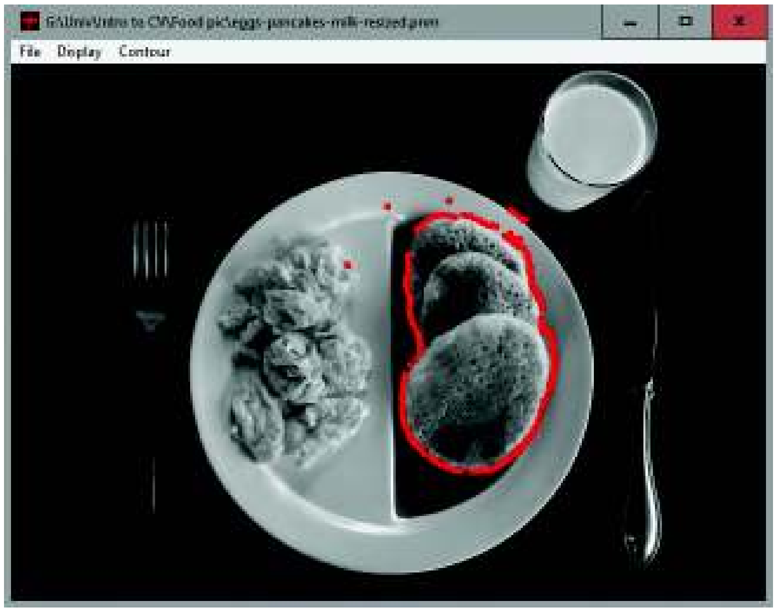

Figure 2: Rubber Band Model - Final Position

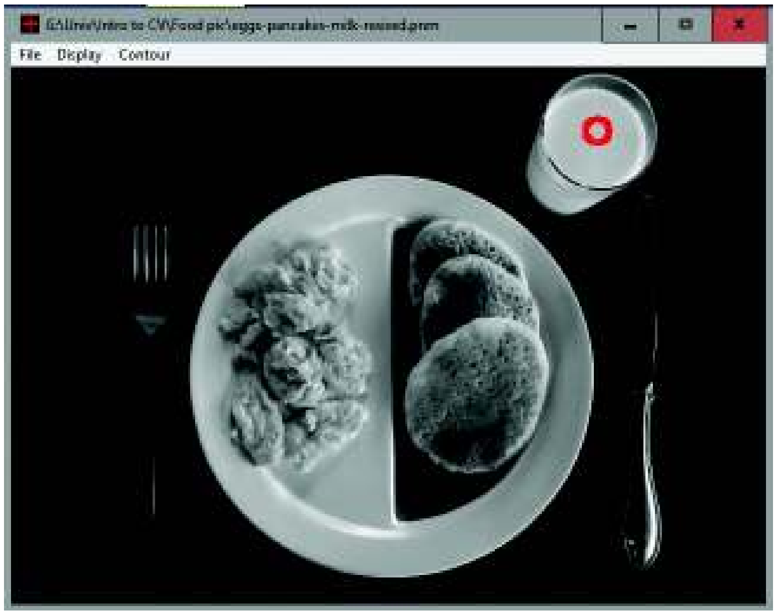

Figure 3: Balloon Model - Final Position

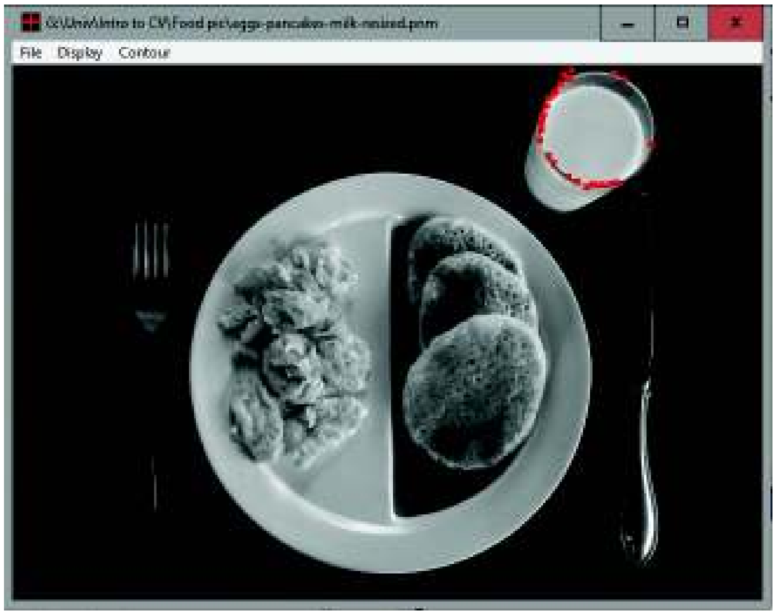

Figure 4: Balloon Model - Final Position

Another option is to add an external energy term based on the color of food objects. This would mean detecting boundaries based on similarity in color. The usage of variance for this did not yield a proper result. An HSV format was used, after conversion from RGB, but using only the Hue values didnot result in any good egdes. Another option is to segment the picture on basis of region growing and use is an external energy term .

4 Conclusions

Thus, an GUI program which allows for semi-automatic segmentation of objects using active contours has been implemented. The final program, though not perfect, thus implement the algoirthm for the active contours well enough to give a satisfactory output. The C source file and the header files used are included in the appendix.

5 Appendix : C Code

5.1 Header file: globals.h

```
#define SQR(x) ((x)*(x)) /* macro for square */
#ifndef M_PI /* in case M_PI not found in math.h */
#define M_PI 3.1415927
#endif
#ifndef M_E
#define M_E 2.718282
#endif

#define MAX_FILENAME_CHARS 320

char filename[MAX_FILENAME_CHARS];

HWND MainWnd;

// Display flags
int ShowPixelCoords;

// Image data
unsigned char *OriginalImage;
int ROWS,COLS;

#define TIMER_SECOND 1 /* ID of timer used for animation */

// Drawing flags
int TimerRow,TimerCol;
int ThreadRow,ThreadCol;
int ThreadRunning;

// Function prototypes
LRESULT CALLBACK WndProc(HWND,UINT,WPARAM,LPARAM);
void PaintImage(unsigned char *);
void AnimationThread(void *); /* passes address of window */

void WindResize(HWND hWnd, int Width, int Height)
{
RECT Client, Wind;
POINT change;
GetClientRect(hWnd, &Client);
GetWindowRect(hWnd, &Wind);
change.x = (Wind.right - Wind.left) - Client.right;
change.y = (Wind.bottom - Wind.top) - Client.bottom;
MoveWindow(hWnd, Wind.left, Wind.top, Width + change.x, Height + change.y, TRUE);
}

//contour gloabls

int contour_on=0;
int count = 0;

struct points
{
int row ;
int col ;
};
struct points *active,*bunch,*new_set,*all;   \\Used for storing active conotur points, init
int bunch_end;       \\points
int RBMLabel = 1; int BalloonLabel = 1;
float *SobelImage;
void Sobel();
void ExternalEnergy(float**);
void InternalEnergyRBM();
void Create7x7Window();
void AssignContour();
void Create7x7Window();
void RBMModel();
void BalloonModel();
void CalTotalE();

unsigned char *OutImage, *red, *blue, *green, *H; unsigned char *intensity;
\\Used for edges based on color
int value = 0;
float *Cx, *Cy;
int completed = 0;
int selected = 0; int move_point; int RBMFlag = 0; int BalloonFlag = 0;
struct points *final; int finalcount = 0; int PNFlag = 0;
```

5.2 Header file: Contour.h

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>

//Sobel Image Calculation
int sobel1[] = { 1, 2, 1, 0, 0, 0, -1, -2, -1 };
int sobel2[] = { -1, 0, 1, -2, 0, 2, -1, 0, 1 };
```

```
void Sobel()
{

SobelImage = (float*)calloc(ROWS*COLS, sizeof(float *));
Cx = (float *)calloc(ROWS*COLS, sizeof(unsigned char*));
Cy = (float *)calloc(ROWS*COLS, sizeof(unsigned char*));
float max, min,max1,max2,min1,min2;min1=min2=max1=max2= max = min = 0; float sum1, sum2, su
float *SobelImage_unnorm;
SobelImage_unnorm = (float*)calloc(ROWS*COLS, sizeof(float*));
for (int i = 3; i < ROWS - 3; i++)
{
for (int j = 3; j < COLS - 3; j++)
{
sum1 = 0; sum2 = 0; sum = 0;
for (int r = 0; r < 3; r++)
{
for (int c = 0; c < 3; c++)
{
sum1 += sobel1[r * 3 + c] * OriginalImage[(i + r - 1)*COLS + j + c - 1] / 9;
sum2 += sobel2[r * 3 + c] * OriginalImage[(i + r - 1)*COLS + j + c - 1] / 9;

}
}Cx[i*COLS + j] = sum1; Cy[i*COLS + j] = sum2;
SobelImage_unnorm[i*COLS + j] = (sum1*sum1 + sum2*sum2);
if (min1 == 0 || Cx[i*COLS + j] < min1)
min1 = Cx[i*COLS + j];
if (max1 < Cx[i*COLS + j])
max1 = Cx[i*COLS + j];
if (min2 == 0 || Cy[i*COLS + j] < min2)
min2 = Cy[i*COLS + j];
if (max2 < Cy[i*COLS + j])
max2 = Cy[i*COLS + j];
if (min == 0 || SobelImage_unnorm[i*COLS + j] < min)
min = SobelImage_unnorm[i*COLS + j];
if (SobelImage_unnorm[i*COLS + j] > max)
max = SobelImage_unnorm[i*COLS + j];
}
}
for (int i = 0; i < ROWS; i++)
{
for (int j = 0; j < COLS; j++)
{
Cx[i*COLS + j] = (Cx[i*COLS + j] - min1) / (max1 - min1);
Cy[i*COLS + j] = (Cy[i*COLS + j] - min2) / (max2 - min2);
SobelImage[i*COLS + j] = (SobelImage_unnorm[i*COLS + j] - min) / (max - min);
}
```

```
}
free(SobelImage_unnorm);
SobelImage_unnorm = NULL;
}

//Egdes Based on color - Comparison with four neighbors
void IntensityEdge()
{
int thresr = 80; int thresb = 80; int thresg =80;
for (int r = 1; r < ROWS-1; r++)
{
for (int c = 1; c < COLS-1; c++)
{
if (abs(red[r*COLS + c] - red[r*COLS + c - 1])>thresr || abs(red[r*COLS + c] - red[r*COLS +
intensity[r*COLS + c] = 1;
else if (abs(blue[r*COLS + c] - blue[r*COLS + c - 1])>thresb || abs(blue[r*COLS + c] - blue[
intensity[r*COLS + c] = 1;
else if (abs(green[r*COLS + c] - green[r*COLS + c - 1])>thresg || abs(green[r*COLS + c] - gr
intensity[r*COLS + c] = 1;
}
}
}

//Calculates External Energy
void ExternalEnergy(float **ExtE)
{
(*ExtE) = (float*)calloc(ROWS*COLS, sizeof(float*));
//Using Sobel Gradient Image and Intensity of Image
float SobelWeight, IntensityWeight,IntEdgeW;
SobelWeight = -1.5; IntensityWeight = 01.5;
if (PNFlag==1)
IntEdgeW = .5;
else
{
IntEdgeW = 0; for (int r = 0; r < ROWS; r++)
for (int c = 0; c < COLS; c++)
intensity[r*COLS + c] = 0;
}

for (int r = 0; r < ROWS; r++)
for (int c = 0; c < COLS; c++)
(*ExtE)[r*COLS + c] = (IntensityWeight*OriginalImage[r*COLS + c] / 255) - SobelWeight*Sobel
}

\\Creates an 7x7 Window
void Create7x7Window(float *input, int r, int c, float **out)
```

```
{
(*out) = (float*)calloc(7*7, sizeof(float*));
int k = 0;
for (int i = -3; i <= 3;i++)
{
for (int j = -3; j <= 3; j++){
if (r + i >= 0&&r+i<ROWS&&c+j>=0&&c+j<COLS)
(*out)[k] = input[(r + i)*COLS + c + j];
else (*out)[k] = 0;

k++;
}
}
}

\\Calculates Curvature
void InternalEnergyCurv(float **IntE, struct points *act, int pointerc, int pointerp, int po
{
(*IntE) = (float*)calloc(7 * 7, sizeof(float*)); int r, c, r1, c1;
unsigned char *window7x7;
float *curve;
curve = (float*)calloc(7 * 7, sizeof(float*));
float sum = 0; float max, min; max = min = 0; int k = 0;
for (int r = -3; r <= 3; r++)
{
for (int c = -3; c <= 3; c++)
{

sum = pow(act[pointern + 1].row - 2 * (act[pointerc].row + r) + act[pointerp].row, 2) + pow(
(*IntE)[k] = sum; k++;
if (min == 0 || min > sum)
min = sum;
if (max < sum)
max = sum;

}
}
for (int g = 0; g < 49; g++)
{
(*IntE)[g] = ((*IntE)[g] - min) / (max - min);
}
}


//Calculates Internal Energy
void InternalEnergy(float **IntE,struct points *act,int pointern)
```

```
{
(*IntE) = (float*)calloc(7*7, sizeof(float*)); int r, c,r1,c1;
unsigned char *window7x7;
float sum = 0; float max, min; max = min = 0; int k = 0;


for (r = -3; r <= 3; r++)
{
for (c = -3; c <= 3; c++)
{
r1 = act[pointern].row + r; c1 = act[pointern].col + c;
sum = 0;
for (int i = 0; i < count; i++)
{
if (i != pointern)
{
sum += pow(r1 - act[i].row, 2) + pow(c1 - act[i].col, 2);
}
}
(*IntE)[k] = sum;
if (min == 0 || min>sum)
min = sum;
if (max < sum)
max = sum;
k++;
}
}
float *curve; int weight;
if (BalloonFlag == 1)
weight = -1;
else if (RBMFlag == 1)
weight = 1;
InternalEnergyCurv(&curve, act, pointern, pointern - 1, pointern + 1);
for (int i = 0; i < 49; i++)
{
(*IntE)[i] = (((*IntE)[i]- min) / (max - min))+weight*curve[i]; \\Curvature+Elasticity
}
}


void CalTotalEBalloon(float *ie, float *ee, int *change_r, int *change_c)
{
float total; float max, min; max = min = 0;
for (int i = 0; i < 7; i++)
{
for (int j = 0; j < 7; j++)
```

```
{
total = +100*ie[i*7 + j] +100*ee[i*7+ j];
if (min == 0 || min>total)
{
min = total;
(*change_r) = i; (*change_c) = j; \\Calculates total energy and finds pixel with minimum ene
}
if (max < total){
max = total;
}
}
}
}

void CalTotalERBM(float *ie, float *ee, int *change_r, int *change_c)
{
float total; float max, min; max = min = 0;
for (int i = 0; i < 7; i++)
{
for (int j = 0; j < 7; j++)
{
total = 10* ie[i * 7 + j]+100*ee[i * 7 + j];
if (min == 0 || min>total)
{
min = total;
(*change_r) = i; (*change_c) = j; \\Calculates total energy and finds pixel with minimum ene
}
if (max < total)
max = total;
}
}
}



void BalloonModel()
{
for (int times = 0; times < 45;times++)
{
float *IE; int j; float *EE; float *EEwindow; int change_r, change_c; float *ie2;
IE = (float*)calloc(7 * 7, sizeof(float*));
ExternalEnergy(&EE); int k;
new_set = (struct points*)calloc(count, sizeof(*new_set));
for (int i = 0; i < count; i++)
{
if (i == count - 1)
```

```
j = 0;
else j = i + 1;
if (i == 0)
k = count - 1;
else k = i - 1;
InternalEnergy(&IE, active, i);
//BalloonIE2(&ie2, active[i], active[j]);
//InternalEnergyCurv(&ie2,active,i,k,j);
Create7x7Window(EE, active[i].row, active[i].col, &EEwindow);
CalTotalEBalloon(IE, EEwindow, &change_r, &change_c);
new_set[i].row = active[i].row + change_r - 3;
new_set[i].col = active[i].col + change_c - 3;
}
PaintImage(OutImage);
HDC hDC = GetDC(MainWnd);
for (int i = 0; i < count; i++)
{
for (int r = -2; r <= 2; r++)
{
for (int c = -2; c <= 2; c++)
{
SetPixel(hDC, new_set[i].col + c, new_set[i].row + r, RGB(255, 0, 0));
//OutImage[(new_set[i].row + r)*COLS + new_set[i].col + c]=
}
}
//SetPixel(hDC,new_set[i].col, new_set[i].row, RGB(255, 0, 0));
}
ReleaseDC(MainWnd, hDC);
active = new_set; \\Updates contour points

}

}


void RBMModel()
{
for (int times = 0; times < 25;times++){
float *IE; int j; float *EE; float *EEwindow; int change_r, change_c;
IE = (float*)calloc(7 * 7, sizeof(float*));
ExternalEnergy(&EE);
new_set = (struct points*)calloc(count, sizeof(*new_set));
for (int i = 0; i < count; i++)
{
if (i == count - 1)
j = 0;
```

```
else j = i + 1;
InternalEnergy(&IE, active,i);
Create7x7Window(EE, active[i].row, active[i].col, &EEwindow);
CalTotalERBM(IE, EEwindow, &change_r, &change_c);
new_set[i].row = active[i].row + change_r - 3;
new_set[i].col = active[i].col + change_c - 3;
}
PaintImage(OutImage);
HDC hDC = GetDC(MainWnd);
for (int i = 0; i < count; i++)
{
for (int r = -2; r <= 2; r++)
{
for (int c = -2; c <= 2; c++)
{ if (c==0&&r==0)
SetPixel(hDC, new_set[i].col + c, new_set[i].row + r, RGB(0, 255, 0));
else
SetPixel(hDC, new_set[i].col + c, new_set[i].row + r, RGB(255, 0, 0));
//OutImage[(new_set[i].row + r)*COLS + new_set[i].col + c]=
}
}
//SetPixel(hDC,new_set[i].col, new_set[i].row, RGB(255, 0, 0));
}
ReleaseDC(MainWnd, hDC);
active = new_set; \\Updates contour points
}
}
```

5.3 Header File: FixedContour.h (For Moving Points)

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>

\\Calculates InternalEnrgy based on fixed point
void InternalEnergyFixed(float **IntE, struct points *act, int pointern)
{
(*IntE) = (float*)calloc(7 * 7, sizeof(float*)); int r, c, r1, c1;
unsigned char *window7x7;
//r = act.row; c = act.col; r1 = next.row; c1 = next.col;
//Create7x7Window(OriginalImage, r, c, &window7x7);
//calculate dist
float sum = 0; float max, min; max = min = 0; int k = 0;


for (r = -3; r <= 3; r++)
```

```
{
for (c = -3; c <= 3; c++)
{
r1 = act[pointern].row + r; c1 = act[pointern].col + c;
sum = 0;
for (int i = 0; i < count; i++)
{
if (i != pointern)
{
sum += pow(r1 - act[i].row, 2) + pow(c1 - act[i].col, 2);
}
}
(*IntE)[k] = sum;
if (min == 0 || min>sum)
min = sum;
if (max < sum)
max = sum;
k++;
}
}
float *curve;
curve = (float *)calloc(49, sizeof(float*));
InternalEnergyCurv(&curve, act, pointern, pointern - 1, pointern + 1);
for (int i = 0; i < 49; i++)
{
(*IntE)[i] = (((*IntE)[i] - min) / (max - min))+curve[i];
}
}


void BalloonModel1(int move_point)
{
for (int times = 0; times < 25; times++)
{
float *IE; int j; float *EE; float *EEwindow; int change_r, change_c; float *ie2;
IE = (float*)calloc(7 * 7, sizeof(float*));
ExternalEnergy(&EE); int k;
new_set = (struct points*)calloc(count, sizeof(*new_set)); new_set = active;
for (int i = move_point-(count/5); i <= move_point+(count/5); i++)
{
if (i == count - 1)
j = 0;
else j = i + 1;
if (i == 0)
k = count - 1;
else k = i - 1;
```

```
InternalEnergyFixed(&IE, active, i);
//BalloonIE2(&ie2, active[i], active[j]);
//InternalEnergyCurv(&ie2,active,i,k,j);
Create7x7Window(EE, active[i].row, active[i].col, &EEwindow);
CalTotalEBalloon(IE, EEwindow, &change_r, &change_c);
new_set[i].row = active[i].row + change_r - 3;
new_set[i].col = active[i].col + change_c - 3;
if (i == move_point)
new_set[i] = active[i]; \\Keeps point moved unchanged while the other points move
}
PaintImage(OutImage);
HDC hDC = GetDC(MainWnd);
for (int i = 0; i < count; i++)
{
for (int r = -2; r <= 2; r++)
{
for (int c = -2; c <= 2; c++)
{
SetPixel(hDC, new_set[i].col + c, new_set[i].row + r, RGB(255, 0, 0));
//OutImage[(new_set[i].row + r)*COLS + new_set[i].col + c]=
}
}
//SetPixel(hDC,new_set[i].col, new_set[i].row, RGB(255, 0, 0));
}
ReleaseDC(MainWnd, hDC);
active = new_set;

}

}

void RBMModel1(int move_point)
{
for (int times = 0; times < 5; times++){
float *IE; int j; float *EE; float *EEwindow; int change_r, change_c;
IE = (float*)calloc(7 * 7, sizeof(float*));
ExternalEnergy(&EE);
new_set = (struct points*)calloc(count, sizeof(*new_set)); new_set = active;
for (int i = move_point-(count/5); i <= move_point-(count/5); i++)
{
if (i == count - 1)
j = 0;
else j = i + 1;
InternalEnergyFixed(&IE, active, i);
Create7x7Window(EE, active[i].row, active[i].col, &EEwindow);
```

```
CalTotalERBM(IE, EEwindow, &change_r, &change_c);
new_set[i].row = active[i].row + change_r - 3;
new_set[i].col = active[i].col + change_c - 3;
if (i == move_point)
new_set[i] = active[i]; \\Keeps point moved unchanged while the other points move
}
PaintImage(OutImage);
HDC hDC = GetDC(MainWnd);
for (int i = 0; i < count; i++)
{
for (int r = -2; r <= 2; r++)
{
for (int c = -2; c <= 2; c++)
{
if (c == 0 && r == 0)
SetPixel(hDC, new_set[i].col + c, new_set[i].row + r, RGB(0, 255, 0));
else
SetPixel(hDC, new_set[i].col + c, new_set[i].row + r, RGB(255, 0, 0));
//OutImage[(new_set[i].row + r)*COLS + new_set[i].col + c]=
}
}
//SetPixel(hDC,new_set[i].col, new_set[i].row, RGB(255, 0, 0));
}
ReleaseDC(MainWnd, hDC);
active = new_set;
}
}
```

5.4 C Source Code: main.c

```
#include <stdio.h>
#include<stdlib.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 "Contour.h"
#include "FixedPointContour.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);
if (!hWnd)
  return(FALSE);

ShowScrollBar(hWnd,SB_BOTH,FALSE);
ShowWindow(hWnd,nCmdShow);
UpdateWindow(hWnd);
MainWnd=hWnd;

ShowPixelCoords=0;

strcpy(filename,"");
OriginalImage=NULL;
ROWS=COLS=0;

InvalidateRect(hWnd,NULL,TRUE);
UpdateWindow(hWnd);

while (GetMessage(&msg,NULL,0,0))
  {
  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;

switch (uMsg)
  {
  case WM_COMMAND:
    switch (LOWORD(wParam))
      {
case ID_CONTOUR_ON:
contour_on = (contour_on + 1) % 2;
//(contour_used) = (unsigned char*)calloc(ROWS*COLS, sizeof(unsigned char*));
if (contour_on==1)
bunch = (struct points*)calloc(ROWS*COLS, sizeof(*active));
if (contour_on == 0)
{
free(final); final = NULL; finalcount = 0; final = (struct points*)calloc(ROWS*COLS, sizeof(
}
PaintImage(OriginalImage);
break;
  case ID_SHOWPIXELCOORDS:
ShowPixelCoords=(ShowPixelCoords+1)%2;
PaintImage(OriginalImage);
break;
  case ID_FILE_LOAD:
if (OriginalImage != NULL)
  {
  free(OriginalImage);
  OriginalImage=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\0PNM files\0*.pnm\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 || strcmp(header, "P6") == 0 ) || BYTES != 255)
  {
  MessageBox(NULL,"Not a PPM (P5 greyscale) image",filename,MB_OK | MB_APPLMODAL);
  fclose(fpt);
  break;
  }
if (strcmp(header, "P5") == 0)
{
PNFlag = 0;
free(OriginalImage); OriginalImage = NULL;
OriginalImage = (unsigned char *)calloc(ROWS*COLS, 1);
header[0] = fgetc(fpt); /* whitespace character after header */
fread(OriginalImage, 1, ROWS*COLS, fpt);
fclose(fpt);
intensity = (unsigned char*)calloc(ROWS*COLS, 1);

}
if (strcmp(header, "P6") == 0)
{
PNFlag = 1;
free(OriginalImage); OriginalImage = NULL;
OriginalImage = (unsigned char *)calloc(ROWS*COLS, 1);
OutImage = (unsigned char *)calloc(ROWS*COLS, 1);
red = (unsigned char *)calloc(ROWS*COLS, 1);
green = (unsigned char *)calloc(ROWS*COLS, 1);
blue = (unsigned char *)calloc(ROWS*COLS, 1);
intensity = (unsigned char*)calloc(ROWS*COLS, 1);
header[0] = fgetc(fpt); /* whitespace character after header */
unsigned char *temp;
temp = (unsigned char*)calloc(ROWS*COLS * 3, 1);
fread(temp, 1, ROWS*COLS * 3, fpt);
fclose(fpt); int k = 0;
for (int i = 0; i < ROWS; i++)
for (int j = 0; j < COLS; j++)
{
OriginalImage[i*COLS + j] = (temp[k] + temp[k + 1] + temp[k + 2]) / 3;
red[i*COLS + j] = temp[k];
blue[i*COLS + j] = temp[k + 1]; //Loads the RGB values and converts them to a greyscale
green[i*COLS + j] = temp[k + 2]; //value, and also stores the Red, Blue, Green values
intensity[i*COLS + j] = 0; //seperately
OutImage[i*COLS + j] = OriginalImage[i*COLS + j];
k = k + 3;
}
}
final = (struct points*)calloc(ROWS*COLS, sizeof(*final));
RBMPoints = (unsigned char*)calloc(ROWS*COLS, sizeof(unsigned char*));
BalloonPoints = (unsigned char*)calloc(ROWS*COLS, sizeof(unsigned char*));
for (int i = 0; i < ROWS;i++)
{
for (int j = 0; j < COLS; j++)
{
RBMPoints[i*COLS + j] = 0;
BalloonPoints[i*COLS + j] = 0;
}
}
SetWindowText(hWnd,filename);
WindResize(hWnd, COLS, ROWS);
PaintImage(OriginalImage);
Sobel();
if (PNFlag==1)
IntensityEdge();

break;

      case ID_FILE_QUIT:
        DestroyWindow(hWnd);
        break;
      }
    break;
  case WM_SIZE:   /* could be used to detect when window size changes */
PaintImage(OriginalImage);
if (contour_on == 1)
{
  HDC hDC = GetDC(MainWnd);
  for (int i = 0; i < finalcount; i++)
  {
  for (int r = -2; r <= 2; r++)
  {
  for (int c = -2; c <= 2; c++)
  {

  SetPixel(hDC, final[i].col + c, final[i].row + r, RGB(255, 0, 0));
  //OutImage[(new_set[i].row + r)*COLS + new_set[i].col + c]=
  }
  }
  //SetPixel(hDC,new_set[i].col, new_set[i].row, RGB(255, 0, 0));
  }
  ReleaseDC(MainWnd, hDC);
}
    return(DefWindowProc(hWnd,uMsg,wParam,lParam));
break;
  case WM_PAINT:
PaintImage(OriginalImage);
    return(DefWindowProc(hWnd,uMsg,wParam,lParam));
break;
  case WM_LBUTTONDOWN:
    return(DefWindowProc(hWnd,uMsg,wParam,lParam));
break;
  case WM_MOUSEMOVE:
//Drawing Rubber Band Conotur points
  if (completed == 1 && contour_on == 1 && (wParam == MK_LBUTTON|| wParam==MK_RBUTTON))
  {
  xPos = LOWORD(lParam);
  yPos = HIWORD(lParam);
  if (xPos >= 0 && xPos < COLS&&yPos >= 0 && yPos < ROWS){
  int j;
  for (int i = 0; i < count; i++)
  {
  if (i == count - 1)
  j = 0;
  else j = i + 1;
  if (xPos >= active[i].col - 2 && yPos >= active[i].row - 2 && xPos <= active[i].col + 2 &&
  {
  move_point = i; selected = 1;
  //MessageBox(NULL, "Sel", filename, MB_OK | MB_APPLMODAL);
  break;
  }
  //else
  //{
//  MessageBox(NULL, "Select Contour Point", filename, MB_OK | MB_APPLMODAL);
 // break;
  //}
  }
  hDC = GetDC(MainWnd);
  int prev_row, prev_col; prev_row = active[move_point].row; prev_col = active[move_point].c
  active[move_point].row = xPos; active[move_point].row = yPos;
  sprintf(text, "%d,%d=>%d     ", xPos, yPos, OriginalImage[yPos*COLS + xPos]);
  TextOut(hDC, 0, 0, text, strlen(text)); /* draw text on the window */
  if (BalloonFlag == 1)
  BalloonModel1(move_point);
  else if (RBMFlag == 1)
  RBMModel1(move_point);
  ReleaseDC(MainWnd, hDC);

  //execute drag contour for xPos and yPos after adding to contour array;
  }
  }
  if (contour_on == 1 && wParam == MK_LBUTTON)
  {
  int filter = 0;
  xPos = LOWORD(lParam);
  yPos = HIWORD(lParam);
  if (xPos >= 0 && xPos < COLS&&yPos >= 0 && yPos < ROWS)
  {
  if (filter % 5 == 0)
  {
  bunch[value].row = yPos; bunch[value].col = xPos;
  value++;
  }
  filter++;
  }
  hDC = GetDC(MainWnd);
  for (int i = -2; i <=2; i++)
  for (int j = -2; j <=2; j++)
  SetPixel(hDC, xPos + i, yPos + j, RGB(255, 0, 0)); /* color the cursor position red */
  ReleaseDC(MainWnd, hDC);
  //count = value;
  }
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 */
SetPixel(hDC,xPos,yPos,RGB(255,0,0)); /* color the cursor position red */
ReleaseDC(MainWnd,hDC);
}
```

```
  }
    return(DefWindowProc(hWnd,uMsg,wParam,lParam));
break;
  case WM_RBUTTONDOWN:
//Draws Balloon Contour Points
  if (contour_on == 1)
  {
  int filter = 0; int value = 0;
  xPos = LOWORD(lParam);
  yPos = HIWORD(lParam);
  int r, c;
  hDC = GetDC(MainWnd);
  for (int d = -10; d <= 10; d++)
  {
  r = yPos + d;
  c = xPos + sqrt(100 - d*d);
  filter++;
  if (filter % 1 == 0){
  // BalloonPoints[r*COLS + c] = value++;
  bunch[value].row = r; bunch[value].col = c;
  value++;
  }
  for (int i = -2; i <=2; i++)
  for (int j = -2; j <= 2; j++)
  SetPixel(hDC, c + i, r + j, RGB(255, 0, 0));
  }
  for (int d = +10; d >=- 10; d--)
  {
  r = yPos + d;
  c = xPos - sqrt(100 - d*d);
  filter++;
  if (filter % 1 == 0){
  bunch[value].row = r; bunch[value].col = c;
  value++;
  }
  for (int i = -2; i <=2; i++)
  for (int j = -2; j <=2; j++)
  SetPixel(hDC, c + i, r + j, RGB(255, 0, 0));
  }
  count = value;
  ReleaseDC(MainWnd, hDC);
  }
  return(DefWindowProc(hWnd, uMsg, wParam, lParam));
  break;
  case WM_KEYDOWN:
//Starts the movement of Rubber Band Model
```

```
  if (wParam == 'r' || wParam == 'R')
  {
  count = value; RBMFlag = 1;
  active = (struct points*)calloc(count, sizeof(*active));
  for (int i = 0; i < count;i++)
  {
  active[i].row = bunch[i].row; active[i].col = bunch[i].col;
  }
 // AssignContour(RBMPoints, RBMLabel, count, &active);
  free(RBMPoints);
  RBMPoints = NULL;
  RBMPoints = (int*)calloc(ROWS*COLS, sizeof(int*));
//  for (int i = 0; i < 10; i++){
  RBMModel();
  completed = 1; int k = 0;
  for (int i = finalcount; i <finalcount+ count; i++)
  {
  final[i] = active[k]; k++;
  }
  finalcount += count;
//  active = new_set;
  //}

  //RBMLabel++;
  PaintImage(OutImage);
  HDC hDC = GetDC(MainWnd);
  for (int i = 0; i < finalcount; i++)
  {
  for (int r = -2; r <= 2; r++)
  {
  for (int c = -2; c <= 2; c++)
  {

  SetPixel(hDC, final[i].col + c, final[i].row + r, RGB(255, 0, 0));
  //OutImage[(new_set[i].row + r)*COLS + new_set[i].col + c]=
  }
  }
  //SetPixel(hDC,new_set[i].col, new_set[i].row, RGB(255, 0, 0));
  }
  ReleaseDC(MainWnd, hDC);

  }
//Starts movement of Balloon Model
  if (wParam == 'b' || wParam == 'B')
  {
  active = (struct points*)calloc(count, sizeof(*active)); BalloonFlag = 1;
  for (int i = 0; i < count; i++)
  {
  active[i].row = bunch[i].row; active[i].col = bunch[i].col;
  }
  //AssignContour(BalloonPoints, BalloonLabel, count, &active);
  free(BalloonPoints);
  BalloonPoints = NULL;
  BalloonPoints = (int*)calloc(ROWS*COLS, sizeof(int*));
 // for (int i = 0; i < 30; i++){
  BalloonModel();
  completed = 1;
  int k = 0;
  for (int i = finalcount; i <finalcount+ count; i++)
  {
  final[i] = active[k]; k++;
  }
  finalcount += count;
  PaintImage(OutImage);
  HDC hDC = GetDC(MainWnd);
  for (int i = 0; i < finalcount; i++)
  {
  for (int r = -2; r <= 2; r++)
  {
  for (int c = -2; c <= 2; c++)
  {

  SetPixel(hDC, final[i].col + c, final[i].row + r, RGB(255, 0, 0));
  //OutImage[(new_set[i].row + r)*COLS + new_set[i].col + c]=
  }
  }
  //SetPixel(hDC,new_set[i].col, new_set[i].row, RGB(255, 0, 0));
  }
  ReleaseDC(MainWnd, hDC);
  //  active = new_set;
  //}
  // _beginthread(BalloonModel, 0, MainWnd);
//  value = 0;
//   BalloonLabel++;

  }
//Announces that no changes will be made to the current contour and a new contour can be im
  if (wParam == 'a' || wParam == 'A'){

  free(active); active = NULL; BalloonFlag = 0; RBMFlag = 0;
  free(bunch); bunch = NULL;
  bunch = (struct points*)calloc(ROWS*COLS, sizeof(bunch));
  value = 0; completed = 0;
  PaintImage(OutImage);
  HDC hDC = GetDC(MainWnd);
  for (int i = 0; i < finalcount; i++)
  {
  for (int r = -2; r <= 2; r++)
  {
  for (int c = -2; c <= 2; c++)
  {
  if (c == 0 && r == 0)
  SetPixel(hDC, final[i].col + c, final[i].row + r, RGB(0, 255, 0));
  else
  SetPixel(hDC, final[i].col + c, final[i].row + r, RGB(255, 0, 0));
  //OutImage[(new_set[i].row + r)*COLS + new_set[i].col + c]=
  }
  }
  //SetPixel(hDC,new_set[i].col, new_set[i].row, RGB(255, 0, 0));
  }
  ReleaseDC(MainWnd, hDC);
  }
  if (wParam == 'i' || wParam == 'I')
  PaintImage(intensity);
if (wParam == 's'  ||  wParam == 'S')
  PostMessage(MainWnd,WM_COMMAND,ID_SHOWPIXELCOORDS,0);   /* send message to self */
if ((TCHAR)wParam == '1')
  {
  TimerRow=TimerCol=0;
  SetTimer(MainWnd,TIMER_SECOND,10,NULL); /* start up 10 ms timer */
  }
if ((TCHAR)wParam == '2')
  {
  KillTimer(MainWnd,TIMER_SECOND); /* halt timer, stopping generation of WM_TIME events */
  PaintImage(OriginalImage); /* redraw original image, erasing animation */
  }
if ((TCHAR)wParam == '3')
  {
  ThreadRunning=1;
  _beginthread(AnimationThread,0,MainWnd); /* start up a child thread to do other work while
  }
  if ((TCHAR)wParam == '4')
  {
  ThreadRunning=0; /* this is used to stop the child thread (see its code below) */
  }
return(DefWindowProc(hWnd,uMsg,wParam,lParam));
break;
  case WM_TIMER:   /* this event gets triggered every time the timer goes off */
```

```
hDC=GetDC(MainWnd);
SetPixel(hDC,TimerCol,TimerRow,RGB(0,0,255)); /* color the animation pixel blue */
ReleaseDC(MainWnd,hDC);
TimerRow++;
TimerCol+=2;
break;
  case WM_HSCROLL:   /* this event could be used to change what part of the image to draw */
PaintImage(OriginalImage);   /* direct PaintImage calls eliminate flicker; the alternative i
    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(OriginalImage);
    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()
else
  CheckMenuItem(hMenu,ID_SHOWPIXELCOORDS,MF_UNCHECKED);
if (contour_on == 1)
CheckMenuItem(hMenu, ID_CONTOUR_ON, MF_CHECKED);
else
CheckMenuItem(hMenu, ID_CONTOUR_ON, MF_UNCHECKED);
DrawMenuBar(hWnd);

return(0L);
}




void PaintImage(unsigned char *input)

{
PAINTSTRUCT Painter;
HDC hDC;
BITMAPINFOHEADER bm_info_header;
BITMAPINFO *bm_info;
```

```
int i,r,c,DISPLAY_ROWS,DISPLAY_COLS;
unsigned char *DisplayImage;

if (input == NULL)
  return; /* no image to draw */

/* Windows pads to 4-byte boundaries.  We have to round the size up to 4 in each dimension,
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]=input[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=
  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 AnimationThread(HWND AnimationWindowHandle)

{
HDC hDC;
char text[300];

ThreadRow=ThreadCol=0;
while (ThreadRunning == 1)
  {
  hDC=GetDC(MainWnd);
  SetPixel(hDC,ThreadCol,ThreadRow,RGB(0,255,0)); /* color the animation pixel green */
  sprintf(text,"%d,%d     ",ThreadRow,ThreadCol);
  TextOut(hDC,300,0,text,strlen(text)); /* draw text on the window */
  ReleaseDC(MainWnd,hDC);
  ThreadRow+=3;
  ThreadCol++;
  Sleep(100); /* pause 100 ms */
  }
}
```

6 Appendix : Test Results

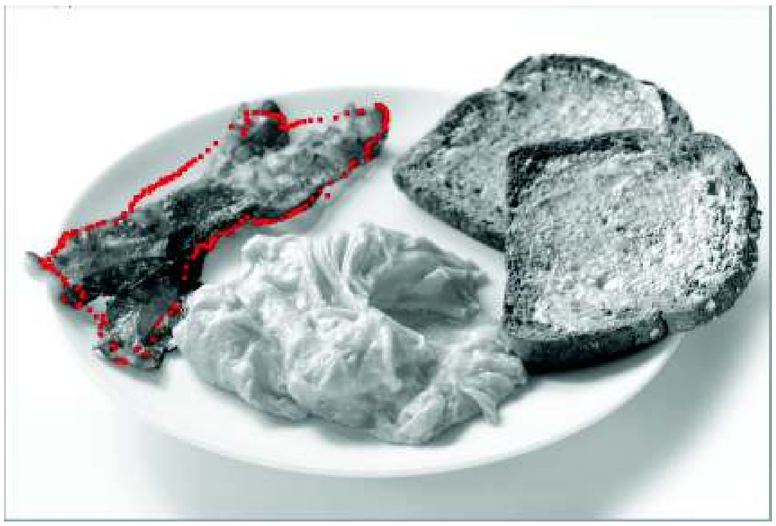

Figure 5: Rubber Band Model - Bacon

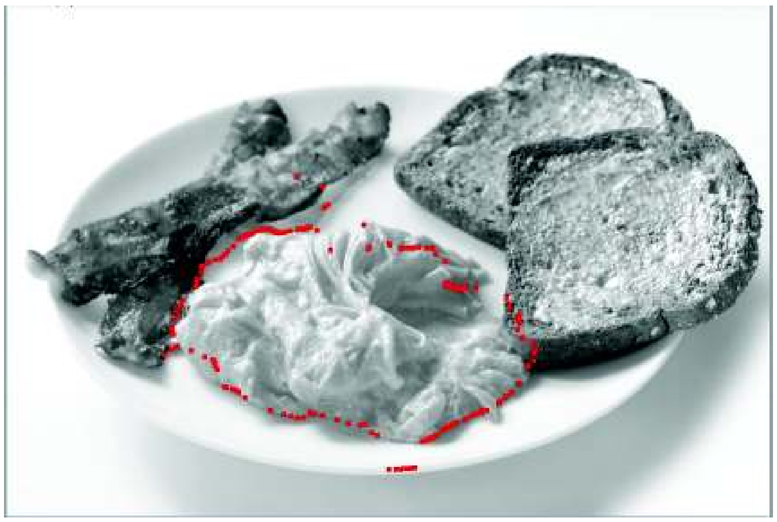

Figure 6: Rubber Band Model - Eggs

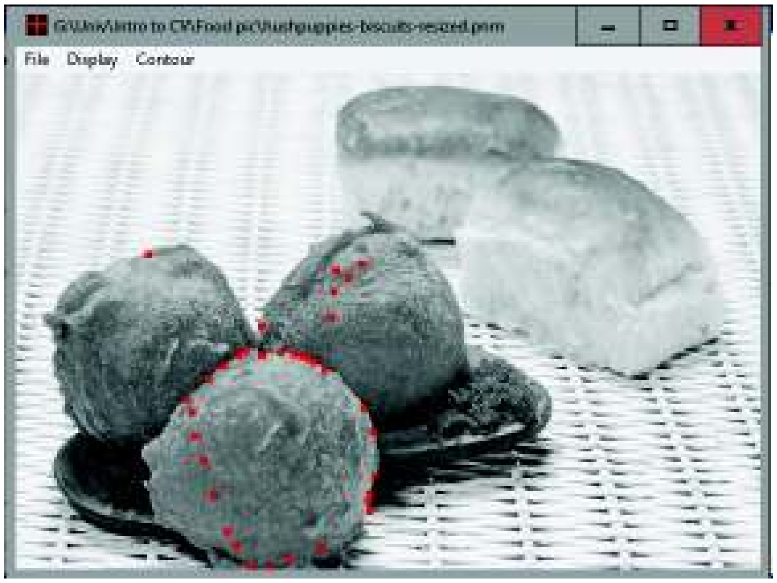
G:\Univ\Intro to CV\Food pic\hushpuppies-biscuits-resized.pnm
File
Display
Contour

Figure 7: Rubber Band Model - Biscuit

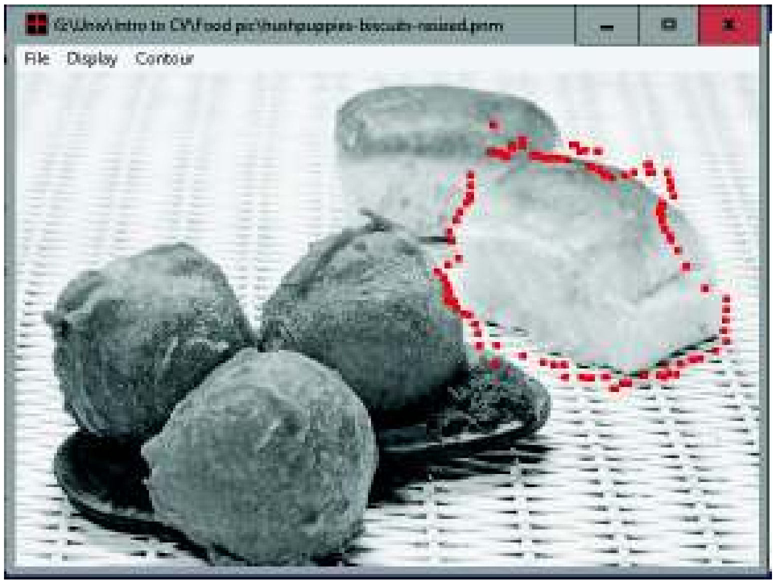
G:\Univ\Intro to CV\Food pic\hushpuppies-biscuits-resized.pnm
File
Display
Contour

Figure 8: Rubber Band Model - Hushpuppies

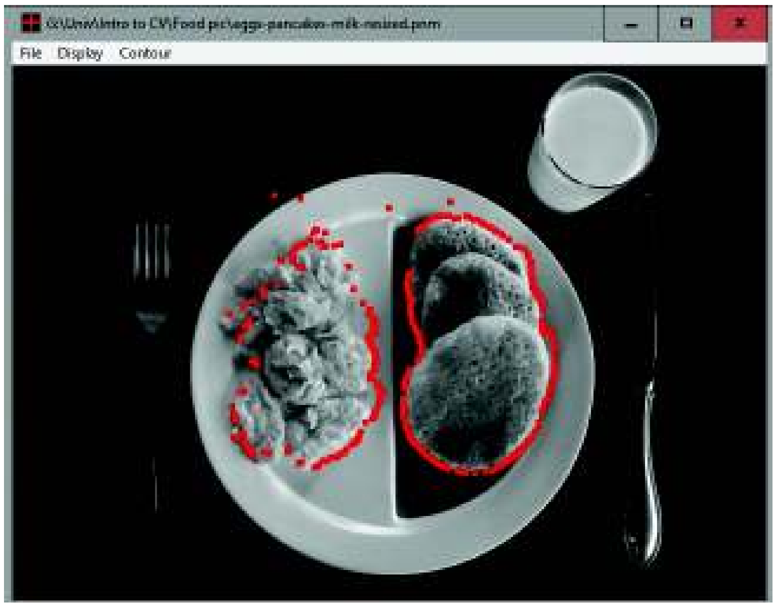

Figure 9: Rubber Band Model - Egg and Pancake

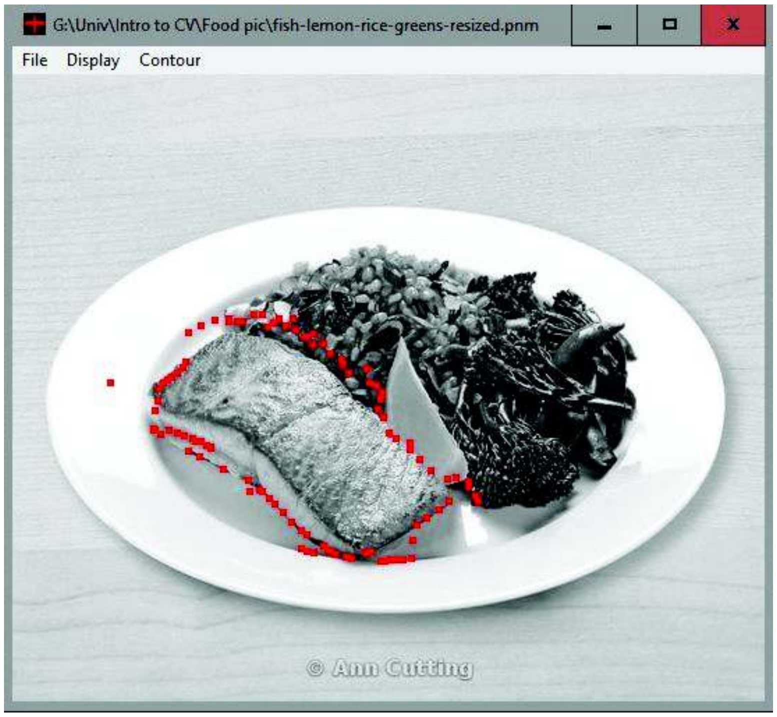
G:\Univ\Intro to CV\Food pic\fish-lemon-rice-greens-resized.pnm
File
Display
Contour
© Ann Cutting

Figure 10: Rubber Band Model - Fish

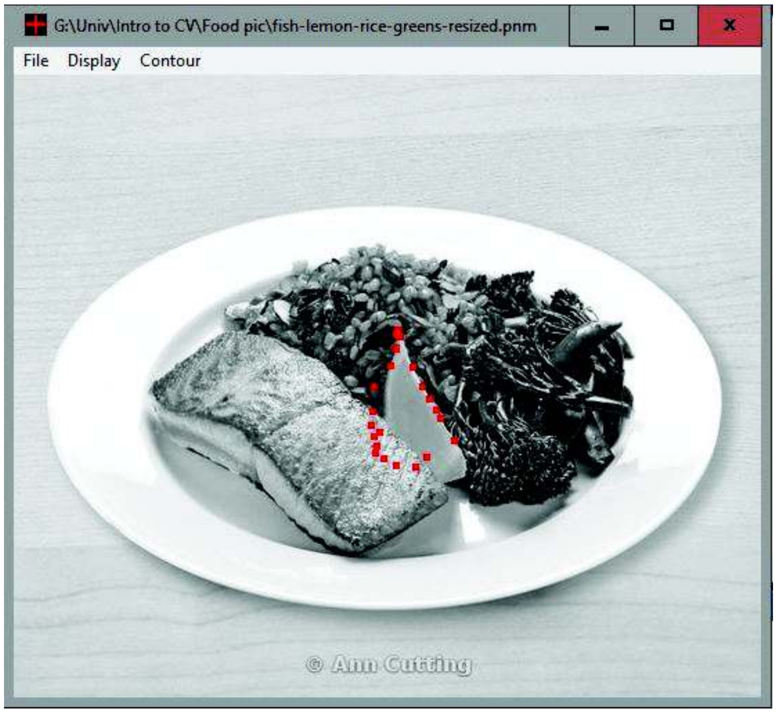
G:\Univ\Intro to CV\Food pic\fish-lemon-rice-greens-resized.pnm
File
Display
Contour
© Ann Cutting

Figure 11: Balloon Model - Lemon

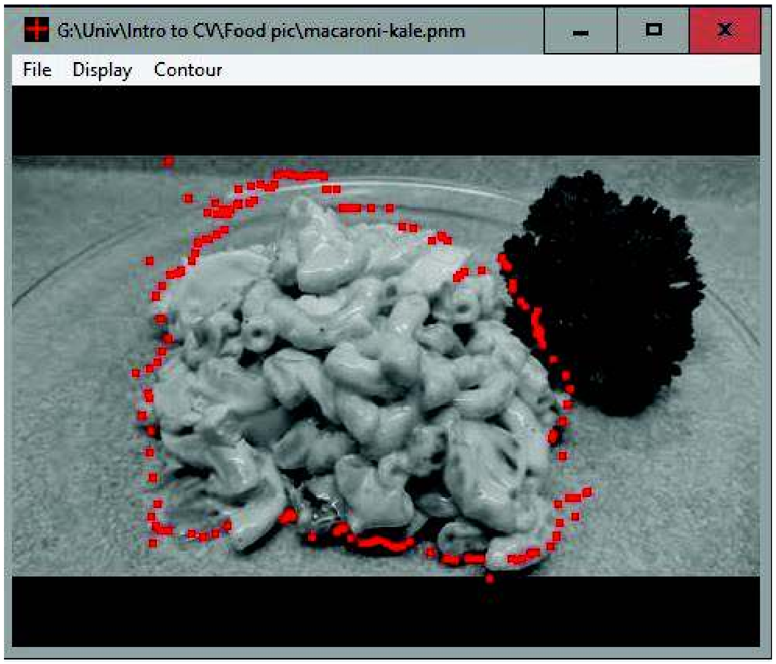

Figure 12: Rubber Band Model - Macaroni