Lab 5: Active Contours

ECE 4310

Brian Guzman

November 5, 2019

Introduction:

The purpose of this lab was to implement an active contours algorithm on a grayscale PPM image. The program was to have two inputs, the original image and a list of X and Y coordinates for the initial contour points. A total of three energies were to be calculated to determine where to move the points to within a 7x7 window. Two internal energies and one external energy. The final contour points were to be displayed as black or white 7x7 '+' shapes.

Energy Calculations:

The first internal energy to be calculated was the energy exerted by the avg distance between all the contour points. To do this a simple distance calculation was done between the current point and the next point. This was done for all points. When dealing with the last point on the list, the "next" point of reference was the first point to create a closed loop. The second internal energy was the difference between the average distance and the distance between the current point and the next point. The third energy term was the external energy exerted by the magnitude of the Sobel Gradient of the input image. To calculate the Sobel gradient the two kernels below were convoluted with the original image:

$$f_1 = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad f_2 = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Figure 1: Sobel Gradient Kernels

The Sobel gradient output had to be normalized to a range of 0 – 255. This was done by finding the max and minimum values and rescaling to the 0 – 255 range. The next step was to normalize the energies in the 7x7 window to a range of 0 – 1. This was done in the same way that the Sobel Gradient image was normalized. Once all energies were normalized the total sum of the energies in the 7x7 window was obtained. Finally, the contour points were moved to the new location. To do this, the minimum energy value within the 7x7 widow was found and the contour point was moved to the new location. This process was repeated 30 times to ensure the best contour positions.

Results/Conclusion:

The first step was to draw the initial contours on the original image. Below is the original image and an image of the initial contours:

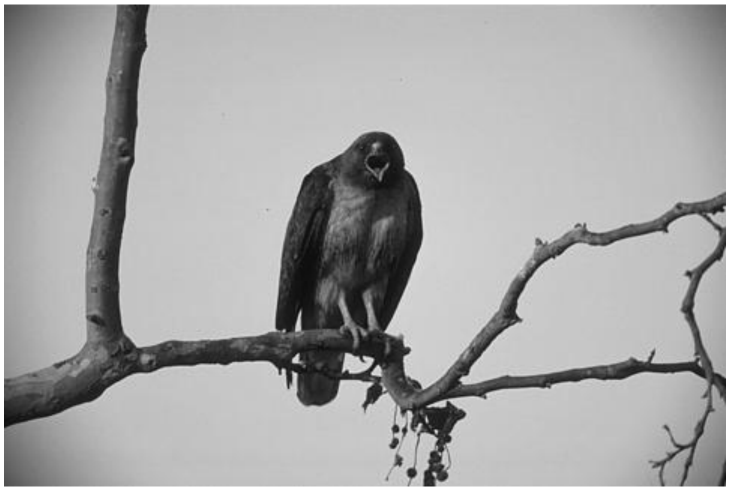
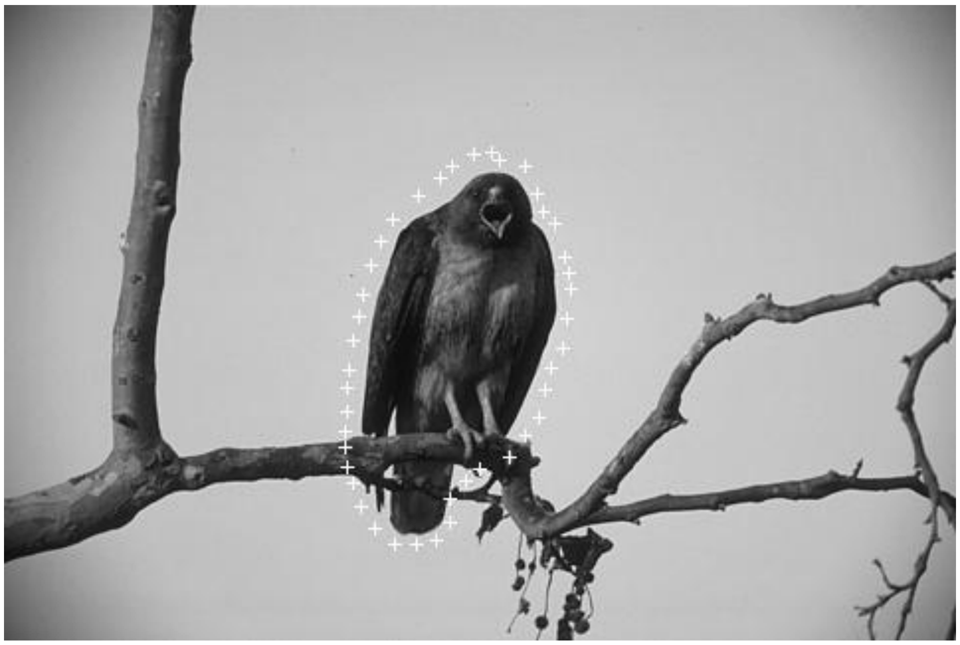


Figure 2: Original Image



Figures 3: Initial Contours

The next step was to get the Sobel gradient to obtain better edges. The images below show the output of the Sobel gradient and its inverse:

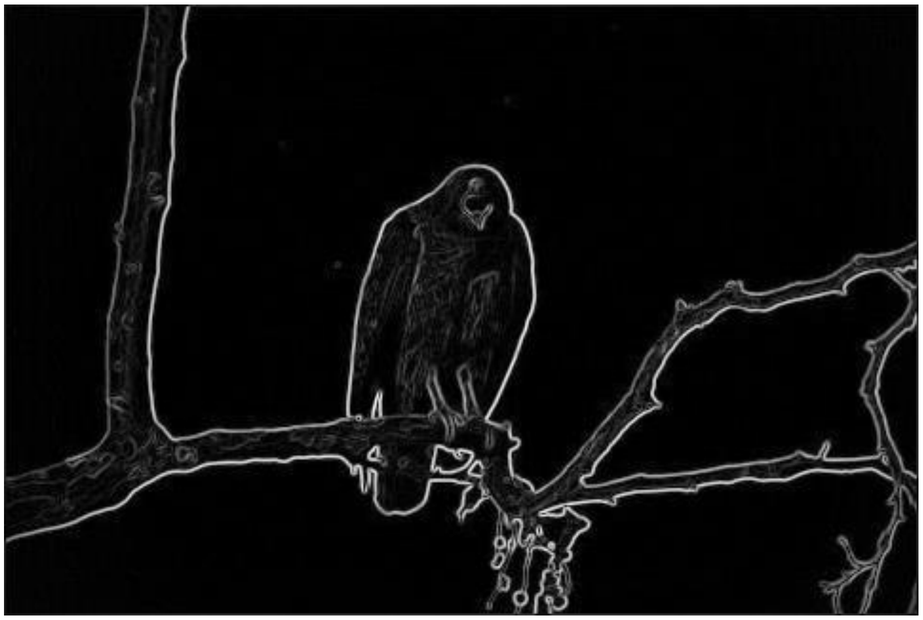


Figure 4: Sobel Gradient Output

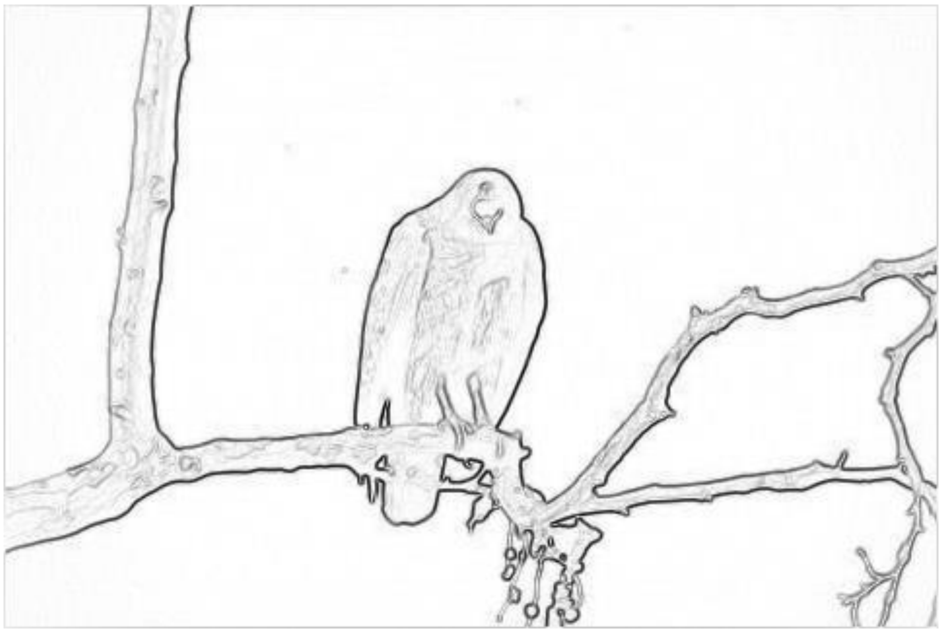


Figure 5: Inverted Sobel Gradient Output

Below is the original image with the final contour points plotted:

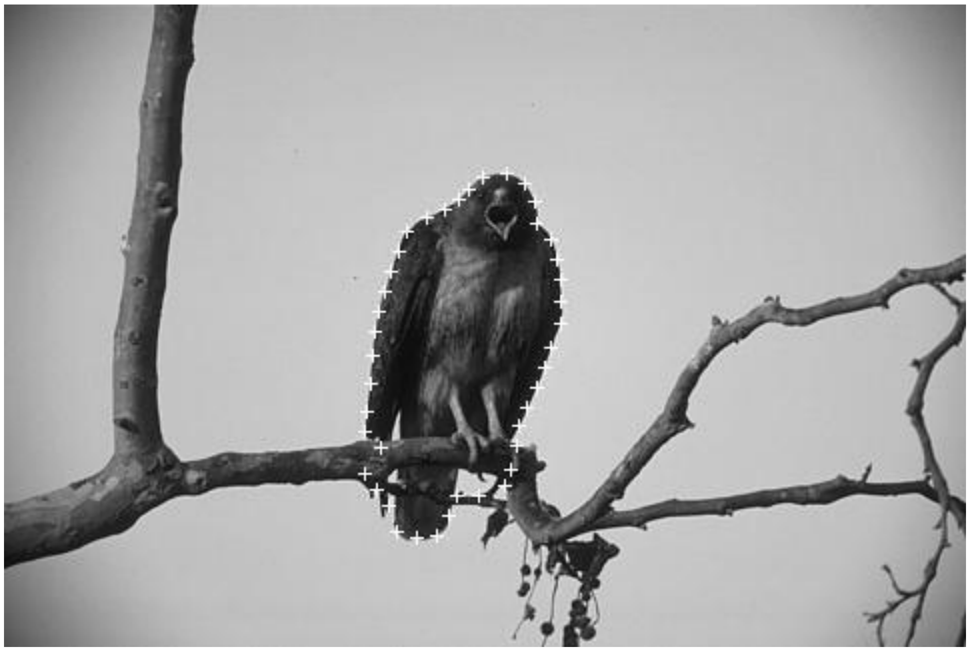


Figure 6: Final Output

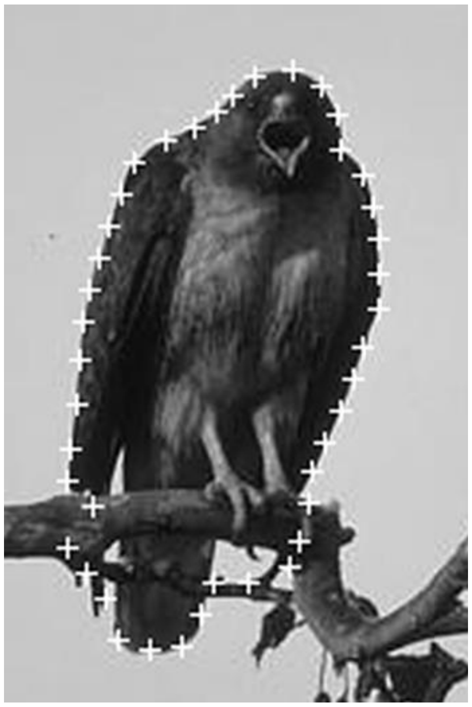


Figure 7: Final Contours

The final contour positions are able to detect the hawk successfully. Some of the points near the branches are not quite on the hawk but this is due to the gradient magnitudes of the branch. These could potentially be adjusted by weighing the internal and external energies a little differently. Weighing allows a certain energy to have more impact on the location of the contour points. In this case, internal energy two (the difference between the average distance and the distance between the current and next points) was amplified by doubling its magnitude when summing up the energies. Before this, some points were being pulled away from the hawk.

Appendix A: New Contour Positions

COLS	ROWS
265	98
266	109
273	117
276	127
278	137
278	148
278	159
273	171
270	181
266	191
261	201
257	210
256	220
253	232
250	240
237	245
226	245
222	255
216	265
206	266
196	263
192	250
186	242
180	234
188	221
180	213
181	203
183	189
184	175
185	163
187	153

190	143
193	133
197	123
201	113
211	106
220	102
227	97
232	92
239	86
251	84
260	89

Source Code:

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


#define MAXRUNS 30
#define SQUARE(x) ((x) * (x))

//READ THE CONTENTS OF A GIVEN FILE
unsigned char *readImageData(FILE *inputImage, int inputR, int inputC)
{
	unsigned char *returnFile;

	returnFile = (unsigned char *)calloc(inputR * inputC, sizeof(unsigned char));
	fread(returnFile, 1, inputR * inputC, inputImage);
	fclose(inputImage);

	return(returnFile);
}

// SOBEL GRADIENT
float *sobelGradient(unsigned char *inputImage, int ROW, int COL)
{
	FILE *outputFile;
	float *gradientImage;
	int *convolutionImage;
	unsigned char *normalizedImage;
	int i, j, r, c;
	int MIN = 0, MAX = 0, x, y, pos, pos1, pos2;

	// KERNELS
	int gx[9] = { -1, 0, 1, -2, 0, 2, -1, 0, 1 };
	int gy[9] = { -1, -2, -1, 0, 0 , 0, 1, 2, 1 };

	convolutionImage = (int *)calloc(ROW * COL, sizeof(int));
	gradientImage = (float *)calloc(ROW * COL, sizeof(float));

	for (i = 0; i < ROW * COL; i++)
	{
		convolutionImage[i] = inputImage[i];
	}

	for (r = 1; r < ROW - 1; r++)
	{
		for (c = 1; c < COL - 1; c++)
		{
			x = 0;
			y = 0;
			for (i = -1; i < 2; i++)
			{
				for (j = -1; j < 2; j++)
				{
					pos1 = ((r + i) * COL) + (c + j);
					pos2 = (3 * (i + 1)) + (j + 1);
					x += (inputImage[pos1] * gx[pos2]);
					y += (inputImage[pos1] * gy[pos2]);
				}
			}
			pos = (r * COL) + c;
			convolutionImage[pos] = sqrt((x * x) + (y * y));
			gradientImage[pos] = sqrt((x * x) + (y * y));
		}
	}

	// FIND MAX AND MIN OF GRADIENT
	MIN = convolutionImage[0];
	MAX = convolutionImage[0];;
	for (i = 1; i < ROW * COL; i++)
	{
		if (MIN > convolutionImage[i])
		{
			MIN = convolutionImage[i];
		}
		if (MAX < convolutionImage[i])
		{
			MAX = convolutionImage[i];
		}
	}

	// NORMALIZE IMAGE
	normalizedImage = (unsigned char *)calloc(ROW * COL, sizeof(unsigned char));
	for (i = 0; i < ROW * COL; i++)
	{
		if (MIN == 0 && MAX == 0)
		{
			normalizedImage[i] = 0;
		}
		else
		{
			normalizedImage[i] = ((convolutionImage[i] - MIN) * (255 - 0) / (MAX 
- MIN)) + 0;
		}
	}
	outputFile = fopen("sobel_output.ppm", "w");
	fprintf(outputFile, "P5 %d %d 255\n", COL, ROW);
	fwrite(normalizedImage, COL * ROW, 1, outputFile);
	fclose(outputFile);

	// INVERT IMAGE
	for (i = 0; i < ROW * COL; i++)
	{
		normalizedImage[i] = 255 - normalizedImage[i];
	}

	outputFile = fopen("inverted_sobel_output.ppm", "w");
	fprintf(outputFile, "P5 %d %d 255\n", COL, ROW);
	fwrite(normalizedImage, COL * ROW, 1, outputFile);
	fclose(outputFile);

	free(normalizedImage);
	free(convolutionImage);

	return(gradientImage);
}

float *normalizeEnergy(float *originalData, int size, float newMin, float newMax, float
min, float max)
{
	float *normalizedOutput;
	int i;

	normalizedOutput = (float *)calloc(size * size, sizeof(float));

	for (i = 0; i < size * size; i++)
	{
		normalizedOutput[i] = ((originalData[i] - min) * (newMax - newMin) / (max -
min)) + newMin;
	}
	return(normalizedOutput);
}

// ACTIVE CONTOURS ALGORITHM
void activeContour(unsigned char *inputImage, float *sobelImage, int inputR, int
inputC,int numContours, unsigned int **contours)
{
	float *internalE1, *internalE2, *externalE, *totalE, *invertedSobel;
	float MIN = 0, MAX = 0, minE;
	float avgX, avgY, avgDistance;
	int i, j, k, contourX, contourY, n;
	int newContourPos[numContours][2];
	int pos, pos2, row, col;
	unsigned char *finalContoursImage;
	int runCount;
	float newMin = 0.0, newMax = 1.0;
	//int debug = 0;

	internalE1 = (float *)calloc(49, sizeof(float));
	internalE2 = (float *)calloc(49, sizeof(float));
	externalE = (float *)calloc(49, sizeof(float));
	totalE = (float *)calloc(49, sizeof(float));
	invertedSobel = (float *)calloc(inputC * inputR, sizeof(float));
	finalContoursImage = (unsigned char *)calloc(inputC*inputR, sizeof(unsigned
char));
```

```
// FIND MIN AND MAX OF SOBEL IMAGE
MIN = MAX = sobelImage[0];
for (i = 1; i < inputR - 1; i++)
{
	for (j = 1; j < inputC - 1; j++)
	{
		pos = (i * inputC) + j;
		if (MIN > sobelImage[pos])
		{
			MIN = sobelImage[pos];
		}
		if (MAX < sobelImage[pos])
		{
			MAX = sobelImage[pos];
		}
	}
}

// INVERT IMAGE
for (i = 0; i < inputC * inputR; i++)
{
	invertedSobel[i] = (float)MAX - sobelImage[i];
}

// ENERGY CALCULATIONS
for (runCount = 0; runCount < MAXRUNS; runCount++)
{
	avgX = avgY = avgDistance = 0.0;

	// AVG DISTANCE BETWEEN POINTS
	for (i = 0; i < numContours - 1; i++)
	{
		avgX = SQUARE(contours[i][0] - contours[i + 1][0]);
		avgY = SQUARE(contours[i][1] - contours[i + 1][1]);

		avgDistance += sqrt(avgX + avgY);
		newContourPos[i][0] = 0;
		newContourPos[i][1] = 0;
	}
	avgX = SQUARE(contours[i][0] - contours[0][0]);
	avgY = SQUARE(contours[i][1] - contours[0][1]);
	newContourPos[i][0] = 0;
	newContourPos[i][1] = 0;

	avgDistance += sqrt(avgX + avgY);
	avgDistance = avgDistance / numContours;

	for (i = 0; i < numContours; i++)
	{
		contourY = contours[i][0]; //col
		contourX = contours[i][1]; //row
		pos = 0;

		for (j = contourX - 3; j <= contourX + 3; j++)
		{
			for (k = contourY - 3; k <= contourY + 3; k++)
			{
				if (i + 1 < numContours)
```

```
                                        {
                                               internalE1[pos] = SQUARE(k - contours[i + 1][0])
+ SQUARE(j - contours[i + 1][1]);
                                               internalE2[pos] = SQUARE(sqrt(internalE1[pos]) -
avgDistance);
                                        }
                                        else
                                        {
                                               internalE1[pos] = SQUARE(k - contours[0][0]) +
SQUARE(j - contours[0][1]);
                                               internalE2[pos] = SQUARE(sqrt(internalE1[pos]) -
avgDistance);
                                        }
                                        pos2 = (j * inputC) + k;
                                        externalE[pos] = SQUARE(invertedSobel[pos2]);
                                        pos++;
                                 }
                          }

                          // MIN AND MAX OF ENERGIES
                          MIN = 0;
                          MAX = 0;
                          float *normInterE1, *normInterE2, *normExt;
                          MIN = MAX = internalE1[0];
                          for (n = 1; n < 49; n++)
                          {
                                 MIN = fmin(MIN, internalE1[n]);
                                 MAX = fmax(MAX, internalE1[n]);
                          }
                          normInterE1 = normalizeEnergy(internalE1, 7, newMin, newMax, MIN,
MAX);
                          MIN = MAX = internalE2[0];
                          for (n = 1; n < 49; n++)
                          {
                                 MIN = fmin(MIN, internalE2[n]);
                                 MAX = fmax(MAX, internalE2[n]);
                          }
                          normInterE2 = normalizeEnergy(internalE2, 7, newMin, newMax, MIN,
MAX);
                          MIN = MAX = externalE[0];
                          for (n = 1; n < 49; n++)
                          {
                                 MIN = fmin(MIN, externalE[n]);
                                 MAX = fmax(MAX, externalE[n]);
                          }
                          normExt = normalizeEnergy(externalE, 7, newMin, newMax, MIN, MAX);

                          // CALCULATE TOTAL ENERGY
                          for (j = 0; j < 49; j++)
                          {
                                 totalE[j] = normInterE1[j] + 2 * normInterE2[j] + normExt[j];
                          }

                          // FIND MINIMUM ENERGY VALUE
                          minE = totalE[0];
                          pos = 0;
                          for (j = 0; j < 49; j++)
                          {
```

```
                        if (minE > totalE[j])
                        {
                                minE = totalE[j];
                                pos = j;
                        }
                }

                // DETERMINE NEW CONTOUR POINTS POSITIONS
                pos2 = pos / 7;
                if (pos2 < 3)
                {
                        // new col = old row - abs(index - 3)
                        newContourPos[i][0] = contours[i][1] - abs(pos2 - 3);
                }
                else if(pos2 > 3)
                {
                        newContourPos[i][0] = contours[i][1] + abs(pos2 - 3);
                }
                else
                {
                        newContourPos[i][0] = contours[i][1];
                }

                pos2 = pos % 7;
                if (pos2 < 3)
                {
                        newContourPos[i][1] = contours[i][0] - abs(pos2 - 3);
                }
                else if (pos2 > 3)
                {
                        newContourPos[i][1] = contours[i][0] + abs(pos2 - 3);
                }
                else
                {
                        newContourPos[i][1] = contours[i][0];
                }

        }

        // MOVE TO NEW POINTS
        for (i = 0; i < numContours; i++)
        {
                contours[i][0] = newContourPos[i][1];
                contours[i][1] = newContourPos[i][0];
        }
}

int x, y;
// DRAW NW CONTOURS
for (i = 0; i < inputC * inputR; i++)
{
        finalContoursImage[i] = inputImage[i];
}
for (i = 0; i < numContours; i++)
{
        y = contours[i][0]; //Col
        x = contours[i][1]; //Row
        for (row = -3; row < 4; row++)
```

```c
			{
				pos = ((x + row) * inputC) + y;
				finalContoursImage[pos] = 255;
			}
			for (col = -3; col < 4; col++)
			{
				pos = (x * inputC) + (y + col);
				finalContoursImage[pos] = 255;
			}
		}

		// WRITE OUTPUT FILE
		FILE *outputFile;
		outputFile = fopen("finalContours.ppm", "w");
		fprintf(outputFile, "P5 %d %d 255\n", inputC, inputR);
		fwrite(finalContoursImage, inputC * inputR, 1, outputFile);
		fclose(outputFile);

		// CREATE FILE WITH FINAL CONTOUR POINTS
		FILE *file;
		file = fopen("contourPoints.csv", "w");
		fprintf(file, "COLS,ROWS\n");
		for (i = 0; i < numContours; i++)
		{
			fprintf(file, "%d,%d\n", contours[i][0],contours[i][1]);
		}
		fclose(file);

		// FREE MEMORY
		free(internalE1);
		free(internalE2);
		free(externalE);
		free(totalE);
		free(invertedSobel);
}


int main(int argc, char *argv[])
{
		FILE *inputImage, *inputTextFile;
		unsigned char *imageData;
		unsigned int **contours;
		int numContourPoints = 0;
		int i;
		char ch;
		int inputMax, inputC, inputR;
		char inputHeader[80];
		unsigned char *initContoursImage;
		float *gradientImage;

		// CHECK USAGE
		if (argc != 3)
		{
			printf("Usage: ./contours <input image> <initial contour file>\n");
			exit(1);
		}

		// OPEN FILES
```

```
	inputImage = fopen(argv[1], "r");
	inputTextFile = fopen(argv[2], "r");
	if (inputTextFile == NULL)
	{
		printf("ERROR: Could not open countours file\n");
		exit(0);
	}
	if (inputImage == NULL)
	{
		printf("ERROR: Could not open input image\n");
		exit(0);
	}

	// PARSE HEADER
	fscanf(inputImage, "%s %d %d %d\n", inputHeader, &inputC, &inputR, &inputMax);

	// READ IMAGE DATA
	imageData = readImageData(inputImage, inputR, inputC);

	// FIND HOW MANY CONTOUR POINTS ARE GIVEN
	while (!feof(inputTextFile))
	{
		ch = fgetc(inputTextFile);
		if (ch == '\n')
		{
			numContourPoints++;
		}
	}
	numContourPoints++; // LAST LINE NOT COUNTED SINCE IT IS THE END OF FILE
	rewind(inputTextFile);

	// INITIALIZE ARRAY TO STORE INITIAL CONTOUR POINT COORDINATES
	contours = (unsigned int **)calloc(numContourPoints, sizeof(unsigned int*));
	for (i = 0; i < numContourPoints; i++)
	{
		contours[i] = (unsigned int *)calloc(2, sizeof(unsigned int));
	}

	//	READ CONTOUR POINTS INTO ARRAY
	for (i = 0; i < numContourPoints; i++)
	{
		fscanf(inputTextFile, "%d", &contours[i][0]);
		fscanf(inputTextFile, "%d", &contours[i][1]);
		//printf("%d %d\n", contours[i][0],contours[i][1]);
	}

	int x, y, pos, row, col;
	// DRAW INITIAL CONTOUR LINE TO IMAGE
	initContoursImage = (unsigned char *)calloc(inputC * inputR, sizeof(unsigned
char));
	for (i = 0; i < inputC * inputR; i++)
	{
		initContoursImage[i] = imageData[i];
	}
	for (i = 0; i < numContourPoints; i++)
	{
		y = contours[i][0]; //Col
		x = contours[i][1]; //Row
```

```
			for (row = -3; row < 4; row++)
			{
				pos = ((x + row) * inputC) + y;
				initContoursImage[pos] = 255;
			}
			for (col = -3; col < 4; col++)
			{
				pos = (x * inputC) + (y + col);
				initContoursImage[pos] = 255;
			}
		}

		// WRITE OUTPUT FILE
		FILE *outputFile;
		outputFile = fopen("initialContours.ppm", "w");
		fprintf(outputFile, "P5 %d %d 255\n", inputC, inputR);
		fwrite(initContoursImage, inputC * inputR, 1, outputFile);
		fclose(outputFile);

		// CALCULATE SOBEL GRADIENT OF IMAGE
		gradientImage = sobelGradient(imageData, inputR, inputC);

		// RUN ACTIVE CONTOURS ALGORITHM
		activeContour(imageData, gradientImage, inputR, inputC, numContourPoints, 
contours);
}
```