Lab 8: Range Image Segmentation

ECE 4310

Brian Guzman

December 4, 2019

The purpose of this lab was to segment a range image based upon surface normal. The image to be tested was a range image of a chair. To implement the segmentation the range image was to be converted from pixels into 3D coordinates using the provided C-code. The 3D coordinates were to be used for calculating the surface normal of a range of pixels. The first step was to take the given range image and threshold it at a certain value. In this case the chose value was 140, this provided the best image to work with and sharper cutoff around the edges. The output of the threshold can be seen in the following images:

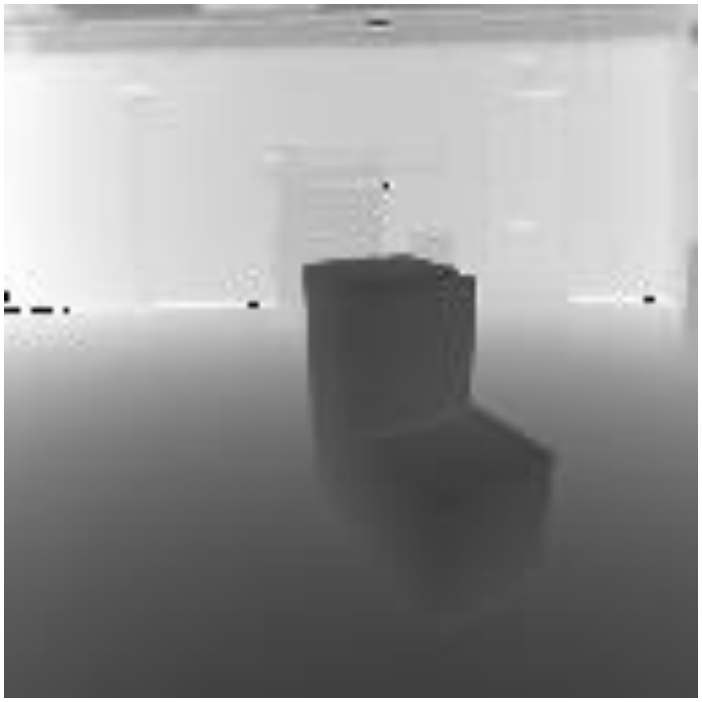

Figure 1: Original Range Image

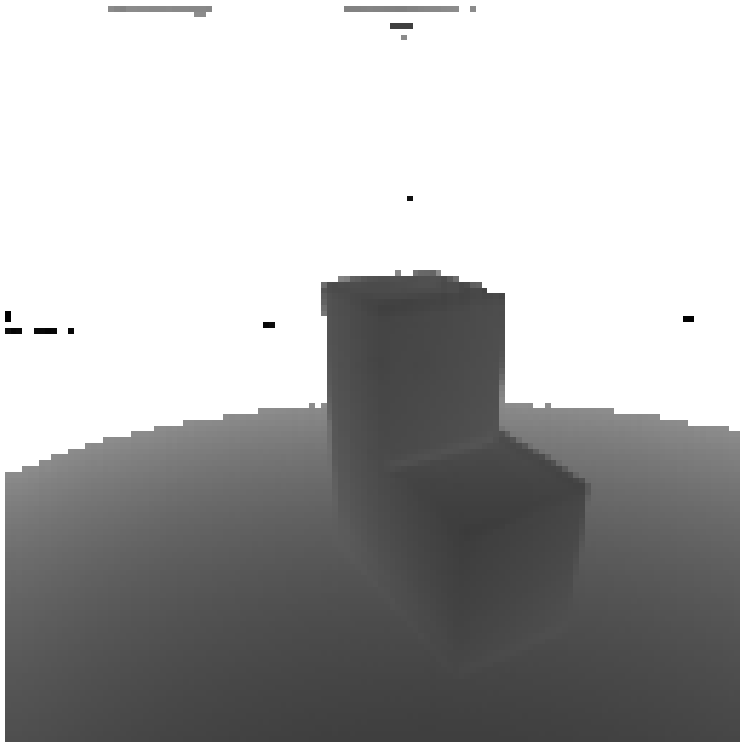

Figure 2: Threshold Output at 140

The next step was to find the surface normal of the point of interest. To do so the cross product of the point of interest was calculated with two other points. These two points were to be at a distance of 3 pixels over from the point of interest. This provided a good distance to where hard cut off edges would be captured. To calculate the surface normal the 3D coordinates were needed so prior to passing the data into the function that calculated the surface normal, the pixels were converted to 3D coordinates. Again, this was done using the provided c-code.

Once the surface normal was calculated, the region growing algorithm was implemented. To do so, the queue-based C code that was provided was to be modified. The conditions for a pixel to be able to join a region were that the pixels angular orientation fell within a certain threshold as those around it. In this case the angular threshold was set to .69 radians. To find the angular difference, the dot product was to be calculated. If a pixel joined an area, the average surface normal was to be recalculated. The condition for a pixel to be a seed pixel was to that it was the only one in a 5x5 window that wasn't already in a region. Once there are no more seed pixels allowed, the region growing algorithm ends. Once all this was calculated, the output final image was obtained. This image contained 7 different regions and looked as follows:

Results:

As can be seen from the image above, the algorithm was able to detect different regions. In the table below we can see the details of each region.

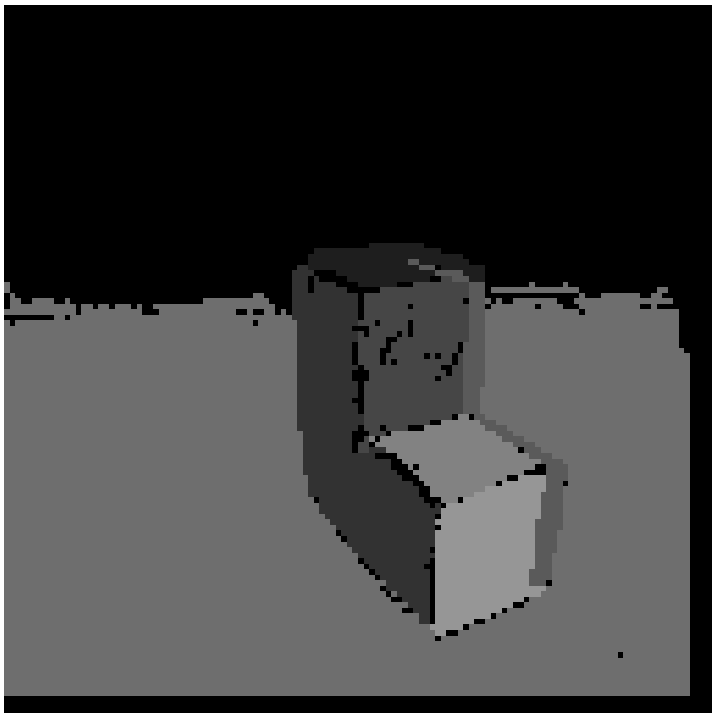

Figure 3: Final Output

Angular Threshold [Rad]	0.69	Image Threshold	140	
Pixel Distance	3	Average Surface Normal		
Region #	# of Pixels	X	Y	Z
1	164	-7.5641	-332.4805	-58.2982
2	753	-51.7945	-1.1391	-8.6044
3	455	2.6062	2.4925	-4.4557
4	209	103.0460	2.4664	-27.4741
5	6730	-4.8922	-28.8855	-8.8800
6	253	-1.1357	-8.5682	-2.3692
7	420	2.7719	1.6290	-4.8366

Table 1: Region Details

Appendix A:

Source Code:

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

#define MAXLENGTH 256
#define THRESHOLD 140
#define PIXEL_WITDH 3
#define MAXQ 10000
#define ANGLETHRESH 0.69
#define SQUARE(x) ((x) * (x))

// STRUCT TO HOLD PIXEL 3D INFO
typedef struct pixel
{
	double x;
	double y;
	double z;
}pixel_t;

// IMAGE READING
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);
}

// THRESHOLD FUNCTION
unsigned char *thresholdImage(unsigned char *imageData, int rows, int cols)
{
	int i;
	unsigned char *output;

	// ALLOCATE MEMORY FOR THREHELD IMAGE
	output = (unsigned char *)calloc(rows * cols, sizeof(unsigned char));
	for (i = 0; i < rows * cols; i++)
	{
		if (imageData[i] <= THRESHOLD)
		{
			output[i] = imageData[i];
		}
		else
		{
			output[i] = 255;
		}
	}
```

```
	// WRITE OUTPUT FILE
	FILE *outputFile;
	outputFile = fopen("thresholdOutput.ppm", "w");
	fprintf(outputFile, "P5 %d %d 255\n", cols, rows);
	fwrite(output, cols * rows, 1, outputFile);
	fclose(outputFile);

	return(output);
}

// PIXEL TO 3D COORDINATE CONVERSION
void pixelTo3D(unsigned char *image, int rows, int cols, pixel_t *pix)
{
	int i, j;
	double x_angle, y_angle, distance;
	double cp[7];
	double slant_correction;
	int index = 0;

	// COORDINATES ALGORITHM
	cp[0] = 1220.7;			/* horizontal mirror angular velocity in rpm */
	cp[1] = 32.0;		/* scan time per single pixel in microseconds */
	cp[2] = (cols / 2) - 0.5;		/* middle value of columns */
	cp[3] = 1220.7 / 192.0;		/* vertical mirror angular velocity in rpm */
	cp[4] = 6.14;		/* scan time (with retrace) per line in milliseconds */
	cp[5] = (rows / 2) - 0.5;		/* middle value of rows */
	cp[6] = 10.0;		/* standoff distance in range units (3.66cm per r.u.) */

	cp[0] = cp[0] * 3.1415927 / 30.0;  /* convert rpm to rad/sec */
	cp[3] = cp[3] * 3.1415927 / 30.0;  /* convert rpm to rad/sec */
	cp[0] = 2.0*cp[0];			/* beam ang. vel. is twice mirror ang. vel. */
	cp[3] = 2.0*cp[3];			/* beam ang. vel. is twice mirror ang. vel. */
	cp[1] /= 1000000.0;			/* units are microseconds : 10^-6 */
	cp[4] /= 1000.0;				/* units are milliseconds : 10^-3 */

	for (i = 0; i < rows; i++)
	{
		for (j = 0; j < cols; j++)
		{
			slant_correction = cp[3] * cp[1] * ((double)j - cp[2]);
			x_angle = cp[0] * cp[1] * ((double)j - cp[2]);
			y_angle = (cp[3] * cp[4] * (cp[5] - (double)i)) + (slant_correction
* 1);  /*  + slant correction */
			index = (i * cols) + j;
			distance = (double)image[index] + cp[6];
			pix[index].z = sqrt((distance * distance) / (1.0 + (tan(x_angle) *
tan(x_angle)) + (tan(y_angle) *tan(y_angle))));
			pix[index].x = tan(x_angle) * pix[index].z;
			pix[index].y = tan(y_angle) * pix[index].z;
		}
	}
}

// SURFACE NORMAL FUNCTION
void findSurfaceNormal(pixel_t *pix, int rows, int cols, pixel_t *surfaceNormal)
{
	int i, j;
```

```
	int pos1, pos2, pos3;
	double x1, x2, y1, y2, z1, z2;

	// CYCLE THROUGH ALL PIXELS MINUS 3 FROM THE RIGHT MOST AND BOTTOM MOST EDGES
	for (i = 0; i < rows - PIXEL_WITDH; i++)
	{
		for (j = 0; j < cols - PIXEL_WITDH; j++)
		{
			// FIND POSITION OF POINT OF INTEREST AND 3 TO THE RIGHT AND 3 BELOW
			pos1 = (i * cols) + j;
			pos2 = ((i + PIXEL_WITDH) * cols) + j;
			pos3 = (i * cols) + j + PIXEL_WITDH;

			// FIND THE VECTORS
			x1 = pix[pos3].x - pix[pos1].x;
			y1 = pix[pos3].y - pix[pos1].y;
			z1 = pix[pos3].z - pix[pos1].z;

			x2 = pix[pos2].x - pix[pos1].x;
			y2 = pix[pos2].y - pix[pos1].y;
			z2 = pix[pos2].z - pix[pos1].z;

			// CALCULATE CROSS PRODUCT
			surfaceNormal[pos1].x = (y1 * z2) - (z1 * y2);
			surfaceNormal[pos1].y = (x1 * z2) - (z1 * x2);
			surfaceNormal[pos1].z = (x1 * y2) - (y1 * x2);
		}
	}
}

// FILL FUNCTION
void qPaintFill(unsigned char *image, unsigned char *outputImage, int ROWS, int COLS, int
r, int c, int paintOverLabel, int newLabel, int *count, pixel_t *pixData)
{
	int r2, c2;
	int q[MAXQ], qh, qt;
	double product;
	double xAvgS, yAvgS, zAvgS;
	double angle = 0;
	double d1 = 0;
	double d2 = 0;
	double total[3] = { 0, 0 ,0 };
	int pos;

	(*count) = 0;
	pos = (r*COLS) + c;

	// AVERAGE IS CURRENT PIXEL VALUE
	xAvgS = pixData[pos].x;
	yAvgS = pixData[pos].y;
	zAvgS = pixData[pos].z;
	q[0] = pos;
	qh = 1;
	qt = 0;

	(*count) = 1;
	total[0] = pixData[pos].x;
	total[1] = pixData[pos].y;
```

```
	total[2] = pixData[pos].z;

	while (qt != qh)
	{
		for (r2 = -1; r2 <= 1; r2++)
		{
			for (c2 = -1; c2 <= 1; c2++)
			{
				pos = (q[qt] / COLS + r2) * COLS + q[qt] % COLS + c2;

				// CONDITIONS THAT MUST BE MET FOR PIXEL TO BE ALLOWED INTO A
REGION
				if (r2 == 0 && c2 == 0)
				{
					continue;
				}
				if ((q[qt] / COLS + r2) < 0 || (q[qt] / COLS + r2) >= ROWS -
PIXEL_WITDH || (q[qt] % COLS + c2) < 0 || (q[qt] % COLS + 2) >= COLS - PIXEL_WITDH)
				{
					continue;
				}
				if (outputImage[pos] != 0)
				{
					continue;
				}

				// DOT PRODUCT OF AVERAGE AND CURRENT PIXEL VALUES
				product = (xAvgS * pixData[pos].x) + (yAvgS * pixData[pos].y)
+ (zAvgS * pixData[pos].z);

				// FIND DISTANCES AND ANGLE OF POINT OF INTEREST
				d1 = sqrt(SQUARE(xAvgS) + SQUARE(yAvgS) + SQUARE(zAvgS));
				d2 = sqrt(SQUARE(pixData[pos].x) + SQUARE(pixData[pos].y) +
SQUARE(pixData[pos].z));
				angle = acos(product / (d1 * d2));

				// IF PIXEL FALLS WITHIN THE THRESHOLD ADD TO REGION
				if (angle > ANGLETHRESH)
				{
					continue;
				}

				(*count)++;
				total[0] += pixData[pos].x;
				total[1] += pixData[pos].y;
				total[2] += pixData[pos].z;
				xAvgS = total[0] / (*count);
				yAvgS = total[1] / (*count);
				zAvgS = total[2] / (*count);
				outputImage[pos] = newLabel;

				q[qh] = (q[qt] / COLS + r2) * COLS + q[qt] % COLS + c2;
				qh = (qh + 1) % MAXQ;
				if (qh == qt)
				{
					printf("Max q size reached\n");
					exit(0);
				}
```

```
                        }
                }
                qt = (qt + 1) % MAXQ;
        }

        //printf("Average Surface Areas\nX: %lf,, Y%lf, Z: %lf\n", xAvgS, yAvgS, zAvgS);
}

int main(int argc, char *argv[])
{
        FILE *inputFile;
        unsigned char *imageData;
        unsigned char *thresheldImage;
        unsigned char *outputImage;
        int ROWS, COLS, MAX;
        char header[80];
        pixel_t *pix, *surfaceNormal;
        int i, j, r, c, p, k;
        int status;
        int count = 0, regionCount = 0;
        int newLabel = 30;

        if (argc != 2)
        {
                printf("ERROR: Usage - ./lab8 <input File>\n");
                exit(1);
        }

        inputFile = fopen(argv[1], "rb");

        if (inputFile == NULL)
        {
                printf("ERROR: Could not open file\n");
                exit(1);
        }

        // READ HEADER
        fscanf(inputFile, "%s %d %d %d\n", header, &COLS, &ROWS, &MAX);

        // READ IMAGE DATA
        imageData = readImageData(inputFile, ROWS, COLS);

        // THRESHOLD IMAGE
        thresheldImage = thresholdImage(imageData, ROWS, COLS);

        // CONVERT PIXEL COORDINATES TO 3D COORDINATES
        pix = (pixel_t *)calloc(ROWS * COLS, sizeof(pixel_t));
        pixelTo3D(imageData, ROWS, COLS, pix);

        // FIND SURFACE NORMAL FOR ALL PIXELS
        surfaceNormal = (pixel_t *)calloc(ROWS * COLS, sizeof(pixel_t));
        findSurfaceNormal(pix, ROWS, COLS, surfaceNormal);

        // REGION GROWING ALGORITHM
        outputImage = (unsigned char *)calloc(ROWS * COLS, sizeof(unsigned char));
        for (i = 2; i < ROWS - 2; i++)
        {
                for (j = 2; j < COLS - 2; j++)
```

```
		{
			status = 1;
			// CHECK 5X5 WINDOW TO SEE IF CURRENT PIXEL CAN BE A SEED
			for (r = -2; r < 3; r++)
			{
				for (c = -2; c < 3; c++)
				{
					p = ((i + r) * COLS) + (j + c);
					if (thresheldImage[p] == 255 || outputImage[p] != 0)
					{
						status = 0;
					}
				}
			}
			// IF PIXEL CAN BE SEED CALL PAINT FILL FUNCTION
			if (status == 1)
			{
				qPaintFill(imageData, outputImage, ROWS, COLS, i, j, 255,
newLabel, &count, surfaceNormal);
				if (count < 100)
				{
					for (k = 0; k < ROWS * COLS; k++)
					{
						if (outputImage[k] == newLabel)
						{
							outputImage[k] = 0;
						}
					}
				}
				else
				{
					regionCount++;
					newLabel += 20;
					printf("Region:\t%d\tNumber of PIxels:\t%d\n",
regionCount, count);
				}
			}
		}
	}
	// WRITE OUTPUT FILE
	FILE *outputFile;
	outputFile = fopen("finalOutput.ppm", "w");
	fprintf(outputFile, "P5 %d %d 255\n", COLS, ROWS);
	fwrite(outputImage, ROWS * COLS, 1, outputFile);
	fclose(outputFile);

	return (0);
}
```