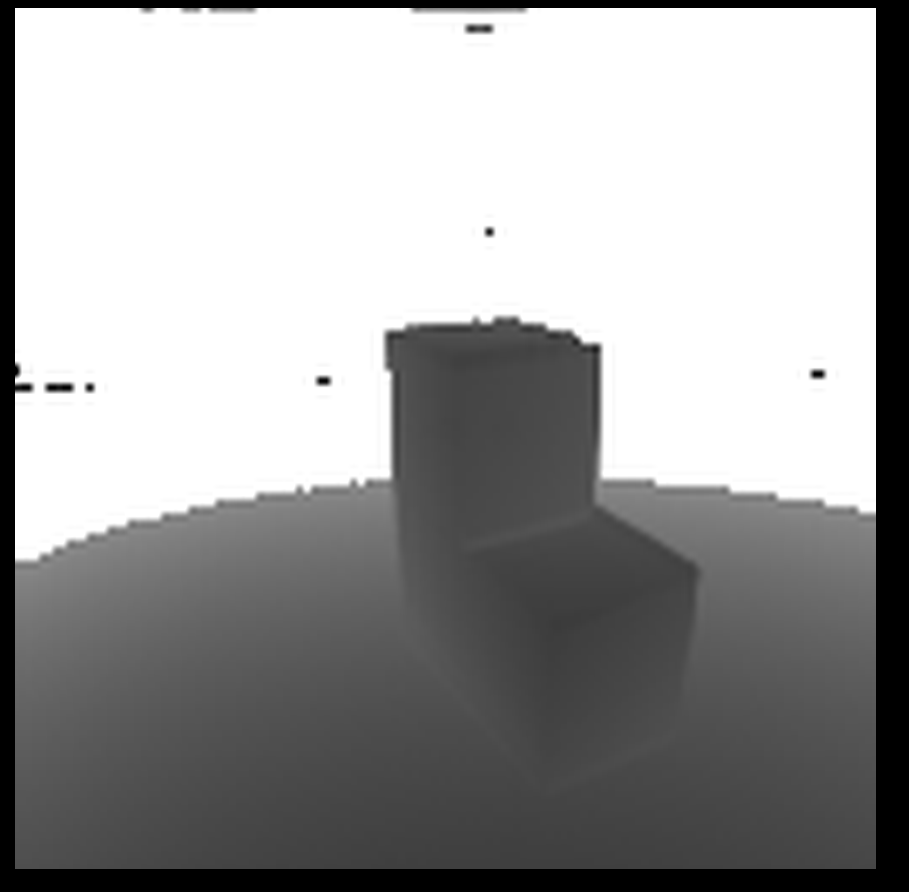
**ECE 4310/6310 Introduction to Computer Vision**

Lab #8 – range Image Segmentation

Name-Pinak Kelkar.

The Below is the output.

Threshold Image -137 which highlights the floor and the chair. Using the code to obtain the X,Y,Z for each pixel.



What we see is the start with a point and look at a windows size of 5x5 which will give us the x,y,z for each points. Now that we know the x,y,z we can save the cross product of each xy,z in that window and add them to the queue to add all of them to the same queue. We do this for all the points and add them to the regions with respect to the cross product of each points. This seed of each region will be added in the process. Each region will be given a label which will help it to identify. As the points in the chair are at 90 degree its easy to region and will be easily visible.



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Region | Number of Pixels: |  |  |  |
| Region: 1 | 163 | -7.982108 | 333.774397 | -58.479712 |
| Region: 2 | 744 | 52.121289 | 1.215533 | -8.658612 |
| Region: 3 | 454 | 2.596830 | -2.493379 | -4.463689 |
| Region: 4 | 205 | 104.818369 | -2.562562 | -27.820564 |
| Region: 5 | 6773 | 1.565296 | 28.785097 | -8.828300 |
| Region: 6 | 243 | 1.227892 | 8.642128 | -2.347285 |
| Region: 7 | 415 | 2.830629 | -1.639295 | -4.864602 |

#include <stdio.h>

#include <stdlib.h>

#include <stdint.h>

#include <string.h>

#include <math.h>

#define THRESHOLD 137

#define ANGLE 0.65

#define MAX\_QUEUE 10000

#define Width 3

// REGION GROW

int Region\_grow(unsigned char \*image, unsigned char \*paint\_image, int rows, int cols,

int current\_row, int current\_col,

int paint\_over\_label, int new\_label,

double \*x, double \*y, double \*z)

{

int count;

int r2,c2;

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

int index;

double dot\_product;

double average\_surface\_X, average\_surface\_Y, average\_surface\_Z;

double angle = 0;

double distance\_A = 0;

double distance\_B = 0;

double total[3] = {0, 0, 0};

count = 0;

index = (current\_row \* cols) + current\_col;

average\_surface\_X = x[index];

average\_surface\_Y = y[index];

average\_surface\_Z = z[index];

total[0] = x[index];

total[1] = y[index];

total[2] = z[index];

queue[0] = index;

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

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

count = 1;

while (qt != qh)

{

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

{

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

{

index = (queue[qt] / cols + r2) \* cols + queue[qt] % cols + c2;

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

{

continue;

}

if ((queue[qt] / cols + r2) < 0 || (queue[qt] / cols + r2) >= rows - Width ||

(queue[qt] % cols + c2) < 0 || (queue[qt] % cols + c2) >= cols - Width)

{

continue;

}

if (paint\_image[index] != 0)

{

continue;

}

dot\_product = (average\_surface\_X \* x[index]) + (average\_surface\_Y \* y[index]) + (average\_surface\_Z \* z[index]);

distance\_A = sqrt( pow(average\_surface\_X, 2) + pow(average\_surface\_Y, 2) + pow(average\_surface\_Z,2) );

distance\_B = sqrt( pow(x[index], 2) + pow(y[index],2 ) + pow(z[index], 2) );

angle = acos(dot\_product / (distance\_A \* distance\_B));

if (angle > ANGLE)

{

continue;

}

count++;

total[0] += x[index];

total[1] += y[index];

total[2] += z[index];

average\_surface\_X = total[0] / count;

average\_surface\_Y = total[1] / count;

average\_surface\_Z = total[2] / count;

paint\_image[index] = new\_label;

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

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

}

}

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

}

printf("X: %lf, Y: %lf, Z: %lf\n", average\_surface\_X, average\_surface\_Y, average\_surface\_Z);

return count;

}

int main ()

{

FILE \*fpt,\*ftr;

unsigned char \*image,\*threshold,\*final;

double \*x,\*y,\*z;

double \*X, \*Y, \*Z;

int \*image\_inverse;

char T\_header[80];

int ROWS,COLS,T\_BYTES;

int r,c;

double cp[7];

double xangle,yangle,dist;

double ScanDirectionFlag=1,SlantCorrection;

double P[3][128\*128];

int ImageTypeFlag;

fpt=fopen("chair-range.ppm","r");

ftr=fopen("chair-reflectance.ppm","r");

int i=0;

int j=0;

i=fscanf(fpt,"%s %d %d %d",T\_header,&COLS,&ROWS,&T\_BYTES);

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

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

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

x=(double \*)calloc(ROWS\*COLS,sizeof(double));

y=(double \*)calloc(ROWS\*COLS,sizeof(double));

z=(double \*)calloc(ROWS\*COLS,sizeof(double));

X=(double \*)calloc(ROWS\*COLS,sizeof(double));

Y=(double \*)calloc(ROWS\*COLS,sizeof(double));

Z=(double \*)calloc(ROWS\*COLS,sizeof(double));

T\_header[0]=fgetc(fpt);

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

fclose(fpt);

///Threhold

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

if (image[i] > THRESHOLD){

threshold[i] = 255;

}

else{

threshold[i] = image[i];

}

}

//////XYZ

fpt=fopen("threshold.ppm","w");

fprintf(fpt,"P5 %d %d 255\n",COLS,ROWS);

fwrite(threshold,1,ROWS\*COLS,fpt);

fclose(fpt);

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

{

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

{

SlantCorrection=cp[3]\*cp[1]\*((double)c-cp[2]);

xangle=cp[0]\*cp[1]\*((double)c-cp[2]);

yangle=(cp[3]\*cp[4]\*(cp[5]-(double)r))+SlantCorrection\*ScanDirectionFlag; /\* + slant correction \*/

dist=(double)image[r\*COLS+c]+cp[6];

z[r\*COLS+c]=sqrt((dist\*dist)/(1.0+(tan(xangle)\*tan(xangle))+(tan(yangle)\*tan(yangle))));

x[r\*COLS+c]=tan(xangle)\*z[r\*COLS+c];

y[r\*COLS+c]=tan(yangle)\*z[r\*COLS+c];

}

}

double x1, x2, y1, y2, z1, z2;

int index1, index2, index3;

for (i = 0; i < (ROWS - Width); i++)

{

for (j = 0; j < (COLS - Width); j++)

{

index1 = (i \* COLS) + j;

index2 = ((i + Width) \* COLS) + j;

index3 = (i \* COLS) + (j + Width);

x1 = x[index3] - x[index1];

y1 = y[index3] - y[index1];

z1 = z[index3] - z[index1];

x2 = x[index2] - x[index1];

y2 = y[index2] - y[index1];

z2 = z[index2] - z[index1];

x[index1] = (y1 \* z2) - (z1 \* y2);

y[index1] = ((x1 \* z2) - (z1 \* x2)) \* -1;

z[index1] = (x1 \* y2) - (y1 \* x2);

}

}

int Flag, regions=0, k;

int new\_label = 30;

int index=0,num=0;;

for (i = 2; i < ROWS -Width; i++)

{

for (j = 2; j < COLS -Width; j++)

{

Flag = 1;

for (r = -2; r < 3; r++)

{

for (c = -2; c < 3; c++)

{

index = ((i + r) \* COLS) + (j + c);

if (threshold[index] == 255 || final[index] != 0)

{

Flag = 0;

}

}

}

if (Flag == 1)

{

num = queue\_paint\_full(image, final, ROWS, COLS, i, j, 255, new\_label, x, y, z);

if (num < 100)

{

for (k = 0; k < (ROWS \* COLS); k++)

{

if (final[k] == new\_label)

{

final[k] = 0;

}

}

}

else

{

regions++;

new\_label += 30;

printf("Region: %d, Number of Pixels: %d\n", regions, num);

}

}

}

}

fpt=fopen("final.ppm","w");

fprintf(fpt,"P5 %d %d 255\n",COLS,ROWS);

fwrite(final,1,ROWS\*COLS,fpt);

fclose(fpt);

}