Shashi Shivaraju
XID:C88650674

REPORT
ECE 6310 Lab #8 – Range Image Segmentation

Objective: To segment a range image based upon surface normals.

Implementation: The segmentation process of the provided range image follows the below steps:

a) Threshold the range image.

A **threshold of pixel intensity `126(INTENSITY_THRESHOLD)`** is used to remove the background and preserve only the floor and chair as seen in the provide range and reflectance images.

[Code snippet]:

```
/*Threshold the image to remove background*/
for(i=0;i<ROW*COL;i++)
{
      if(INTENSITY_THRESHOLD < image[i])
      {
            image[i] = 0;
      }
}
```

b) Convert the pixels of the range image into 3D coordinates.

The provided C code to convert the pixels of the range image to its corresponding 3D coordinates is integrated and the 3D coordinates were stored with scan direction as downward.

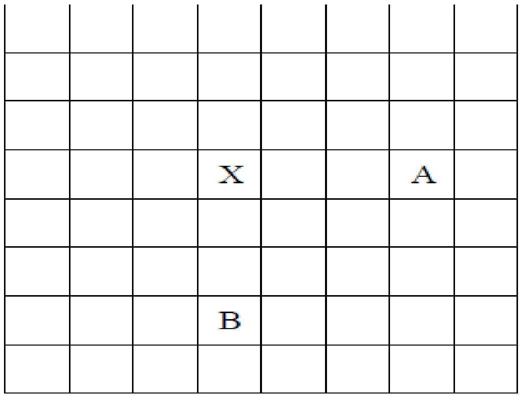
[Code snippet]:

```
double              *coords[3] = {0,0,0};
/*Allocate for 3D Coordinates*/
for(i=0;i<3;i++)
{
      coords[i] = (double  *)calloc(ROW*COL,sizeof(double ));
}
/*Convert pixels into 3D coordinates*/
odetics2coords(image,ROW,COL,coords,0);

/*Function to convert pixels into 3D coordinates*/
void odetics2coords(unsigned char *RangeImage,int ROWS,int COLS,
                        double **P,int ImageTypeFlag);
```

Shashi Shivaraju
XID:C88650674

c) Calculate the surface normals for the image.

Using the obtained 3D coordinates, the surface normal for each pixel is calculated by cross product. The surface normal at pixel X is calculated by taking the cross product of (B-X) x (A-X), where A, B and X are the 3D coordinates of those pixels. **The distance chosen between pixels for cross products is 3 pixels** i.e. the distances B-X and A-X are selected as 3.



[Code snippet]:

```
/*calculate the surface normals for the image */
for(i=0;i<ROW*COL;i++)
{
      /*position of pixel X in image*/
      convert_index2height_width(i,ROW,COL,&xpos,&ypos);
      x[0]=coords[0][i];
      x[1]=coords[1][i];
      x[2]=coords[2][i];

      /*position of pixel A in image*/
      convert_height_width2index(&index,ROW,COL,xpos+NORM_DIST,ypos);
      a[0]=coords[0][index];
      a[1]=coords[1][index];
      a[2]=coords[2][index];

      /*position of pixel B in image*/
      convert_height_width2index(&index,ROW,COL,xpos,ypos+NORM_DIST);
      b[0]=coords[0][index];
      b[1]=coords[1][index];
      b[2]=coords[2][index];

      for(j=0;j<3;j++)
      {
            bx[j]=b[j]-x[j];
            ax[j]=a[j]-x[j];
      }

      /*norm(x) = cross product (b-x)*(a-x)*/
      norms[0][i] = bx[1]*ax[2]-ax[1]*bx[2];
      norms[1][i] = bx[2]*ax[0]-ax[2]*bx[0];
      norms[2][i] = bx[0]*ax[1]-ax[0]*bx[1];

}
```

Shashi Shivaraju
XID:C88650674

d) Segmentation of the range image using region growing.

Queue-based C code for region growing is used to segment regions.

The seed pixels for region growing are found by identifying a complete 5x5 window of unlabeled (and not masked out) region. If any pixel within the 5x5 window is masked out or already labelled in a region, then the pixel is not considered as a seed pixel for the new region. The predicate for the pixel to join the region is that its orientation should be within a threshold of the average orientation of pixels already in the region. The angular difference is calculated using the dot product and the average is recalculated after every new pixel joins the region. **The angle threshold chosen is 40 degrees (`#define ORIENTATION_THRESHOLD 40 /*(Degrees)*/`).**

[Code snippet]:

```
labelcount = 0; /*label for region growing*/
for(R=2;R<ROW-2;R++)
{
      for(C=2;C<COL-2;C++)
      {
            seedpixel_flag = E_TRUE;      /*Assume its a seed pixel*/
            /*Check 5x5 window to ensure it is a valid seed pixel*/
            for(r1=-2;r1<=2;r1++)
            {
                  for(c1=-2;c1<=2;c1++)
                  {
                        if(0 == image[(R+r1)*COL+(C+c1)]||
                              0 != labels[(R+r1)*COL+(C+c1)])
                        {
                              seedpixel_flag = E_FALSE;
                              break;
                        }
                  }
                  if(seedpixel_flag == E_FALSE)
                        break;
            }
            if(seedpixel_flag == E_FALSE)
                        continue;
            labelcount ++;

            /*Use region growing with the seed pixel*/
            RegionGrow(image,labels,ROW,COL,R,C,0, labelcount,
      indices,&RegionSize,norms,ORIENTATION_THRESHOLD);

            /*paint the grown region with gray shades*/
            for (i=0; i<RegionSize; i++)
            {
                  seg_image[indices[i]]= GREY_SHADE+(count-1)*30;
                  /*To Calculate averge surface normals for each region */
                  x[0] = x[0]+norms[0][indices[i]];
                  x[1] = x[1]+norms[1][indices[i]];
                  x[2] = x[2]+norms[2][indices[i]];
            }
        }
 }
```

Shashi Shivaraju
XID:C88650674

[Region Growing Code with predicate formulae highlighted in blocks]:

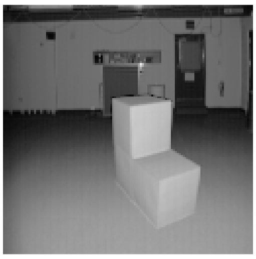
```
void RegionGrow(unsigned char *image,     /* image data */
                        unsigned char *labels,  /* segmentation labels */
                        int ROWS,int COLS,      /* size of image */
                        int r,int c,            /* pixel to paint from */
                        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 */
                        double **norms,    /*3D norms of the image*/
                        int Orientation_Predicate)

{
      int   r2,c2;
      int   queue[MAX_QUEUE],qh,qt;
      double      average,total;    /* average and total orinetation in growing region */
      int pixel_x_pos = 0,pixel_y_pos = 0,index;
      double seed_norm[3] = {0,0,0};
      double pixel_norm[3] = {0,0,0};

      double dot_ab=0,mag_a=0,mag_b=0,cos_theta,theta;
      int x_pos = 0,y_pos = 0;

      *count=0;
      if (labels[r*COLS+c] != paint_over_label)
            return;
      labels[r*COLS+c]=new_label;
      average=total=0;
      convert_height_width2index(&index,ROWS,COLS,c,r);
      /*Seed value*/
      seed_norm[0] = norms[0][index];
      seed_norm[1] = norms[1][index];
      seed_norm[2] = norms[2][index];

      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)
      {
            /* recalculate average after each pixels join */
                  if(*count != 1 )
                  average=total/(*count - 1);

            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;
                        if(image[(queue[qt]/COLS+r2)*COLS+queue[qt]%COLS+c2] == 0)
```

Shashi Shivaraju
XID:C88650674

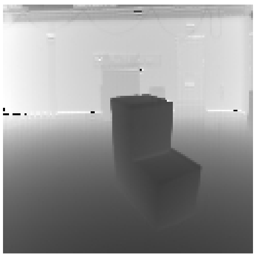
```
                                                        continue;


        /* Orientation test criterias to join region */
        /*Pixel norm value*/
        pixel_norm[0] = norms[0][(queue[qt]/COLS+r2)*COLS+queue[qt]%COLS+c2];
        pixel_norm[1] = norms[1][(queue[qt]/COLS+r2)*COLS+queue[qt]%COLS+c2];
        pixel_norm[2] = norms[2][(queue[qt]/COLS+r2)*COLS+queue[qt]%COLS+c2];

        /*Calculate the dot product and find the orientation*/
        dot_ab=seed_norm[0]*pixel_norm[0]+seed_norm[1]*pixel_norm[1]+seed_norm[2]*pixel_norm[2];
        mag_a=sqrt(SQR(seed_norm[0])+SQR(seed_norm[1])+SQR(seed_norm[2]));
        mag_b=sqrt(SQR(pixel_norm[0])+SQR(pixel_norm[1])+SQR(pixel_norm[2]));
        cos_theta = dot_ab/(mag_a*mag_b);
        theta = CONV_2_DEGREE(acos(cos_theta));

        /*Predicate*/
        if (abs(theta-average) > Orientation_Predicate)
                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 = total + theta;

                                (*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;
        }
}
```

Shashi Shivaraju
XID:C88650674

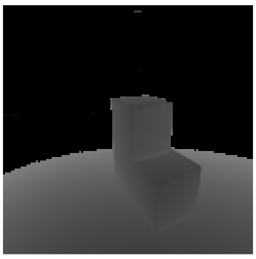
[Reflectance Image]:
128 x 128 8bit Gray scale reflectance image of PPM format named chair-reflectance.ppm



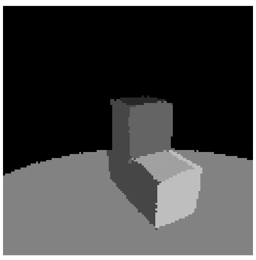
[Input Range Image]:
128 x 128 8bit Gray scale range image of PPM format named chair-range.ppm



[Input range image with threshold as pixel intensity 126]:
128 x 128 8bit Gray scale range image of PPM format named thresholded_image.ppm

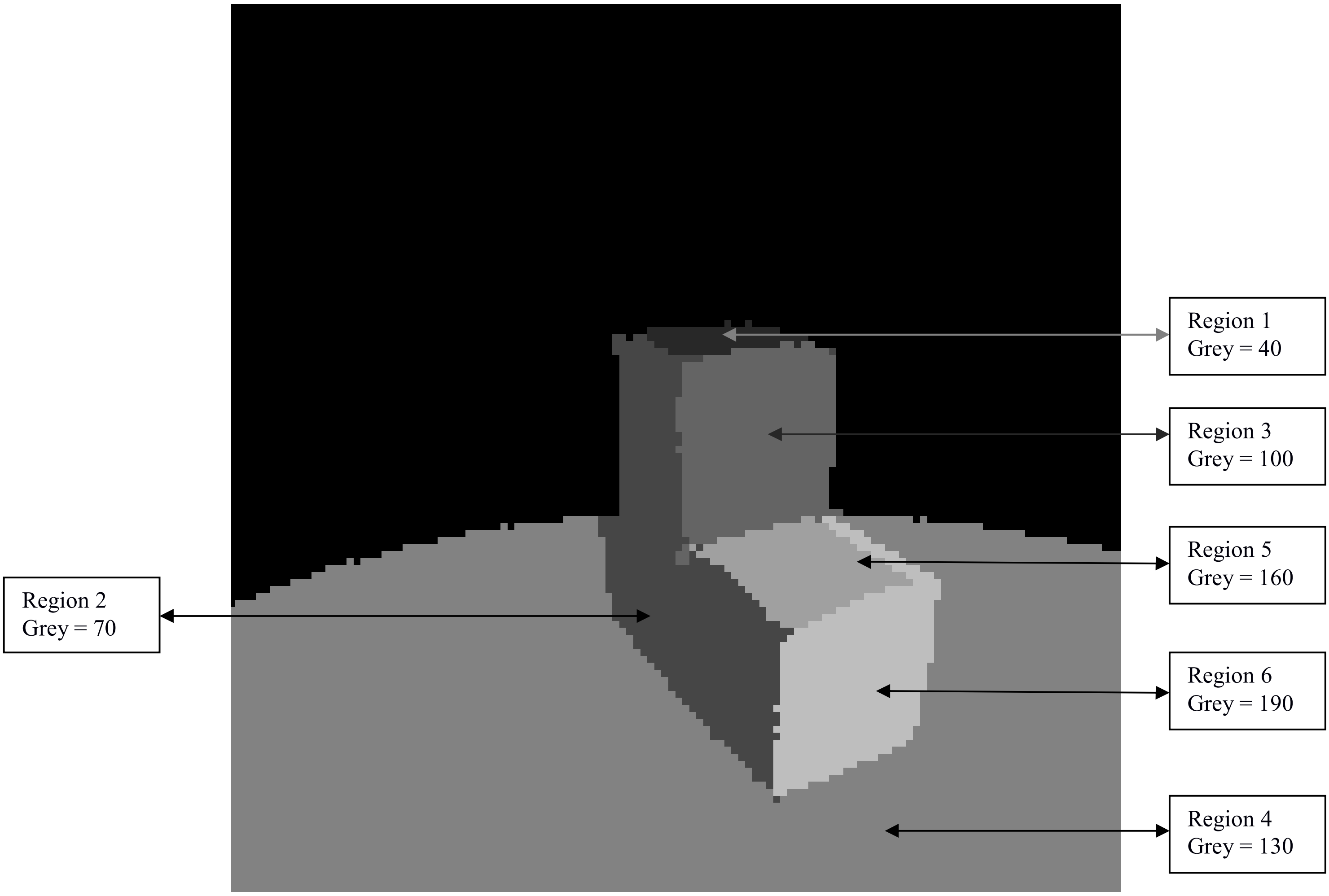


[Output segmented range image]:
128 x 128 8bit Gray scale range image of PPM format named segmented_range_image.ppm



Shashi Shivaraju
XID:C88650674

[Enlarged output segmented range image with regions labeled with grey shade values]:
128 x 128 8bit Gray scale range image of PPM format named thresholded_image.ppm



[Table]:

Region Label	Region Size	Greyscale Representation	Average Surface Normals		
			x	y	z
1	61	40	2.35362	-18.8632	6.426522
2	762	70	12.58546	0.573014	4.712715
3	494	100	-3.3766	2.247218	4.630016
4	5166	130	0.670635	-9.68987	3.108285
5	259	160	0.910146	-8.44529	2.40573
6	532	190	-10.2393	1.505498	7.249829