3D Object Recreation from Multiple Camera Views

By: Anil Rohatgi

Computer Vision

April 27, 2005

Georgia Institute of Technology

Project Goal

The ability to recreate a real object inside a virtual space is vastly becoming a necessity for cutting edge computer vision. The ability to do this operation passively and in real time opens many doors for a wide range of applications. This paper will focus on an approach devised to passively capture information about a desired object, process this data, and recreate the object in a virtual environment. The system implements multiple camera views to use stereo disparity as a means of recovering depth information about the surface of the object at varying angles. The images are passed into a PC and processed so that correlating points in each image can be recovered. Using these points, Matlab can then reconstruct a full 3d model of the desired object.

The Theory

The governing principal behind a multiple camera system for 3D recreation is stereo disparity. Stereo disparity theory relates correlating points from the image projection in multiple cameras to the distance from the observer to the point in space. Using this technique and prior knowledge about the setup, a three dimensional model of the object surface facing the cameras can be developed. If this technique is implemented with multiple cameras facing different sides of the object, depths to the objects surface can be generated on all sides. When these side surfaces are re-combined in a Cartesian plane, the full 360 degree surface of the object can be modeled. A figure of a stereo disparity system for a single point can be seen below:

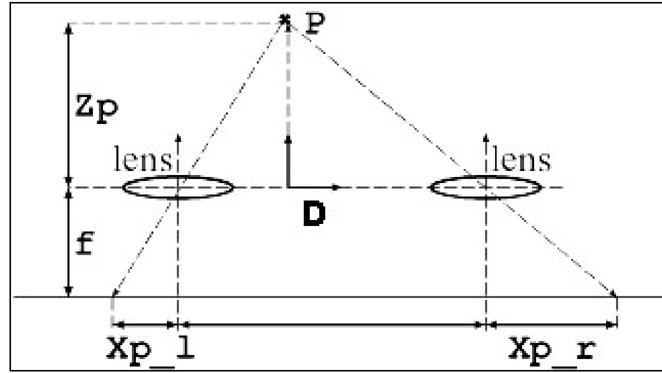

Figure 1 Stereo disparity setup for a single point

In reference to the image above, the point P(x,y,z) corresponds to an point in space.

D is the distance between the cameras, and f is the effective focal length of the camera lens. Xp_1 and Xp_2 are the pixel locations of the image projection of point P from each view. Using this information, the depth to the objects surface at each point, Zp can be determined using equation 1.

Zp=(D*F)/(Xp_l-Xp_r) Eq.1

It should be noted that the depth of the object is inversely proportional to the difference in the x position of the projected point in the multiple images. Assuming a system of correlation is applied, this equation can be extended to use on multiple points on the object and essentially sample the surface of the object facing the camera into discreet three dimensional points. Using this technique of sampling and plotting for multiple camera orientations around the object, a 360 degree surface model can be interpolated from these discreet points.

The Set Up

In order to implement this technique, a system similar to the one described above was constructed. Two cameras were carefully placed at set distance apart, while still maintaining view of the object. These two cameras were placed on a level plane, so that there was no displacement in the vertical direction. This allowed for equal heights in both images, and essentially limited the vertical displacement of corresponding points. The object itself was placed on a rotating surface in view of both cameras. The need for a rotating platform stems from the lack of access to many cameras. This interaction with the object violates the goal of passive capture, but given more equipment this same technique can be employed without touching the object. Instead of rotating the object in respect to the camera's field of view, eight stationary cameras could be placed surrounding the object and set to capture images from all four sides. A projector was attached to a computer, and an image was projected upon the surface of the object facing the cameras. This image was used for correlation purposes described later in the paper. A diagram of the setup described above is included below:

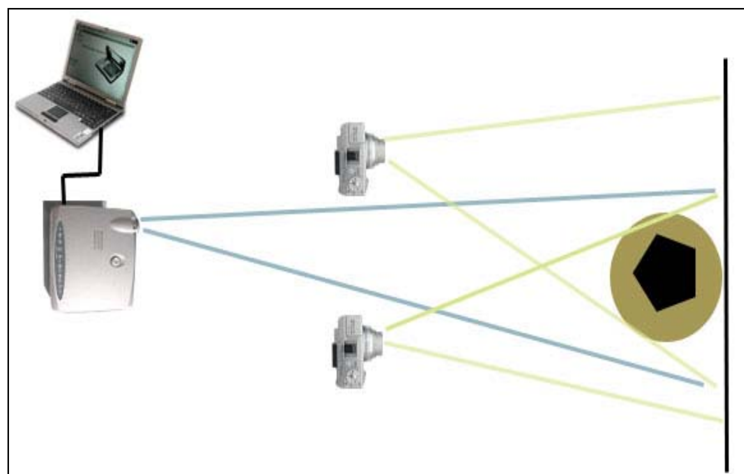

Figure 2. Physical setup of capture system

Point Correlation

Once the images are captured and sent into the computer for processing, they need to be manipulated so that the computer can automatically correlate discreet points from one image to the other. The key to this feature correspondence lies within the image projected upon the object during the capture.

The original idea was to project a color gradient upon the surface of the captured object. What this projected image would do, is assign a unique color value to each vertical column of the object. In this situation, the computer could match color values from one image to another along a horizontal row of pixels. The limitation to the same row of pixels for corresponding points stems from the level plane on which the cameras were placed. Assuming equal framing of both images, with no vertical displacement between cameras, it can be assumed that correlating points from each image lie on the same row of pixels. The discreet resolution of the final image ideally would only be limited by the number of colors the computer could display, and the pixel resolution of the camera. However, limitations in the equipment used for the capture prevented this method of projection from effectively working. The projector could not display the full range of colors the computer could output, and because the change of color in the gradient was so small, the cameras could not distinguish discreet points. Also this high dependence upon color values made the system very susceptible to shadow and noise.

The method that was implemented for the test capture was a projection of a checkerboard pattern on the desired capture object. The drastic change in color between squares showed up nicely on camera, and allowed for simpler feature extraction. The original captured images for the test are shown below.

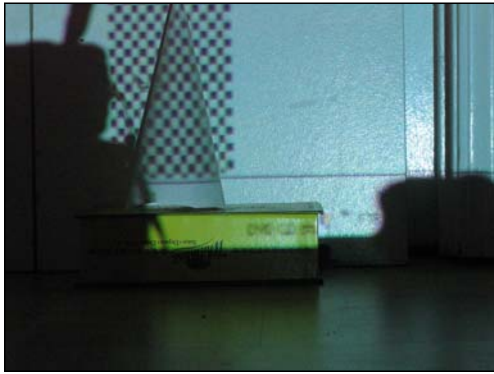

Figure 3. View from right camera

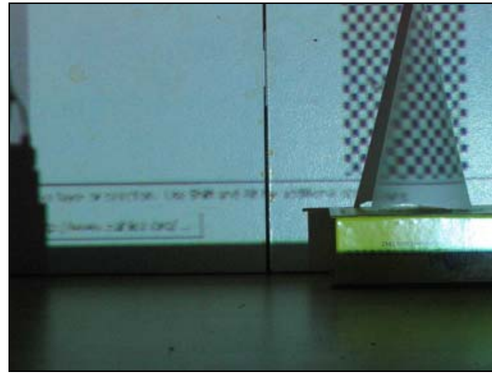

Figure 4. View from left camera

The pictures were then cropped to fit the boundary of the checker board pattern projection. This served two purposes. One, to ensure the images truly contained the same region of data, and two, in order to make it easier to see the changes in corresponding checkers from one image to another.

Image Filtering

Even though the checker pattern is easily viewable from the images presented above, the noise in the camera made it difficult for the computer to see where a checker started and stopped. Also, shadows created on the outer edge of the cone further complicated the checker boundary extraction. It is for this reason that further filtering was needed to make it easier for the computer to see where a vertical checker column starts and stops. The first filter that was applied was a saturation filter, used to generate a more drastic change in color values between the black and the white checkers. However, the noise in the image caused a fuzzy region along the checker boundary, still making it difficult for the computer to extract the checker boundary. To try and alleviate this boundary noise, a

median smoothing filter was applied in order to lower the noise content in the image. The corresponding output images are shown below.

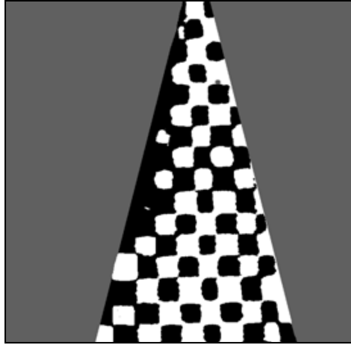

Figure 5. Filtered right image.

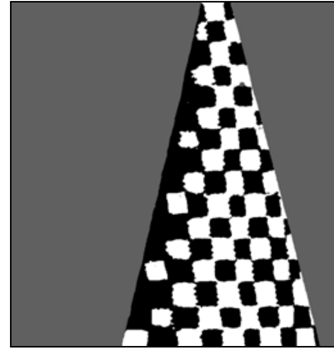

Figure 6. Filtered left image.

From Checkers to Points

Now that the computer can clearly determine a distinction between the squares of the checker board, it is time to convert these checker boundaries into correlating points. To do this, a single point of correlation needed to be entered manually, and based upon this one point, all other points of correlation can be determined. For use in the test, a fold in the paper could be seen in the third column from the right in both images. By outlining the column of checkers in line with this point, the computer can take over and locate the boundaries of corresponding checkers. The algorithm to do this works as follows.

The coordinates of the column manually entered are stored in two arrays, one of x values and one of y values. For each y value in the array, the algorithm traverses the image from the corresponding x value to the right of the image, logging each drastic change in color as a checker boundary. This result is stored in a matrix, with the height index corresponding to the y value, and the width index corresponds to the pixel index of each edge detected from left to right. The algorithm then traverses the image from the corresponding x value to the left of the image, again logging each checker boundary in a

matrix. In the end these matrices are compiled into one giant matrix, that holds the x value of each checker for a given y value. A flow diagram of this algorithm is shown below.

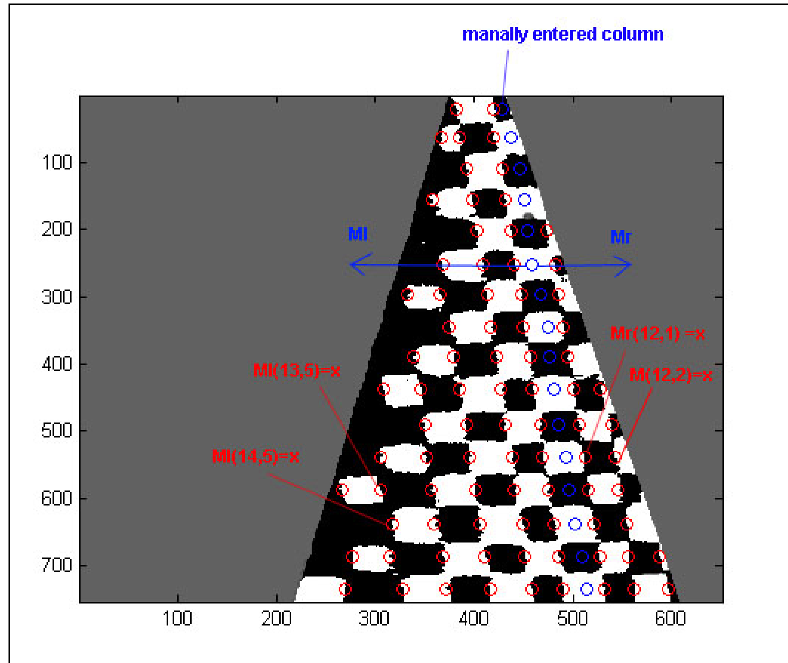

Figure 7. Flow diagram for point correspondence

Mr in the diagram corresponds to the matrix containing the edge pixel values going outward from the manually selected points to the right of the image boundary. Ml likewise, corresponds to the matrix containing edge pixel values going outward from the manually entered points to the left of the image. The first index in either of these matrices corresponds to the checker row number. The second index number of these matrices is the column number. The value stored in the matrix is the x value of the pixel of the edge. The fact that the traversal starts outward from a known point of correspondence ensures that each element in the matrix corresponds to the same element in the similar matrix for the other image. The Mr and Ml matrices are concatenated into one image matrix. The resulting matrix for the right image and the left image are seen below.

0	0	0	0	348	383	0	0	0
0	0	0	334	343	382	0	0	0
0	0	0	0	350	387	0	0	0
0	0	0	321	353	388	0	0	0
0	0	0	0	355	392	430	0	0
0	0	0	318	359	392	434	0	0
0	0	292	318	360	395	438	0	0
0	0	0	322	363	395	439	0	0
0	0	287	324	364	400	441	0	0
0	0	0	328	369	400	445	482	0
0	0	289	328	368	405	446	483	504
0	251	290	331	374	407	449	487	0
0	255	293	332	373	409	453	488	519
0	252	294	335	379	410	456	495	0
207	249	297	339	378	416	456	494	526
208	256	300	339	384	416	460	500	529

Figure 8. Right image edge matrix

0	0	0	0	382	420	0	0	0
0	0	0	367	385	419	0	0	0
0	0	0	0	392	428	0	0	0
0	0	0	358	398	432	0	0	0
0	0	0	0	403	438	474	0	0
0	0	0	368	409	441	482	0	0
0	0	333	366	413	448	485	0	0
0	0	0	375	417	450	490	0	0
0	0	338	379	422	457	495	0	0
0	308	346	385	427	459	500	528	0
0	0	350	392	432	468	507	539	0
0	305	352	395	439	469	512	542	0
266	305	356	401	441	475	516	546	0
0	317	360	406	449	481	521	555	0
277	314	369	411	451	486	527	556	588
269	328	372	416	459	490	532	562	597

Figure 9. Left image edge matrix

It should also be noted that an attempt to interpolate a smooth surface between these discreet surface points was made by finding a linear regression to the boundaries of each checker column. However, because these lines extended indefinitely, it was too difficult to truncate them into fitting the size and shape of the reference object. This method was discarded. Images of the interpolation are included below.

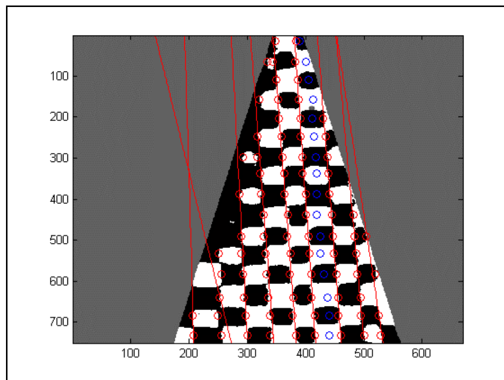

Figure 10. Linear regression right

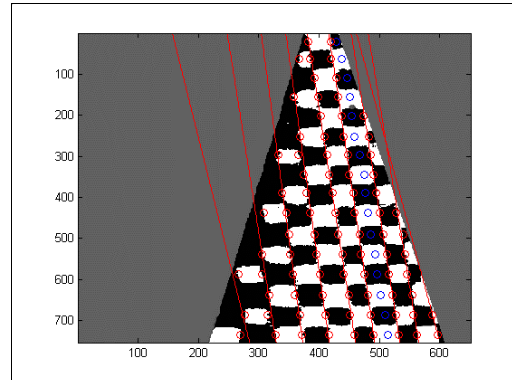

Figure 11. Linear regression left

Object Reconstruction

Now that a matrix of correlating points and their x position values exists, calculating the disparity between each point is done by merely taking the difference between the matrix of each image. The result of this operation is a depth map of the

object. However, some points exist in only one image, so this subtraction leads to a very large false value. To prevent this, a mask of zeros and ones was created for each image. These image masks were multiplied together, and then multiplied with the disparity map to eliminate these unusually high values. The resultant matrix is ready to plot.

For the first attempt at a surface plot, the MESH function in Matlab was used and the result of this operation can be seen below. The figure showed a clear conical trend. However, this was only for one side of the object and contained the zero depth levels plotted for regions that were not part of the object. It is for these reasons that combining multiple side surfaces with this function was difficult.

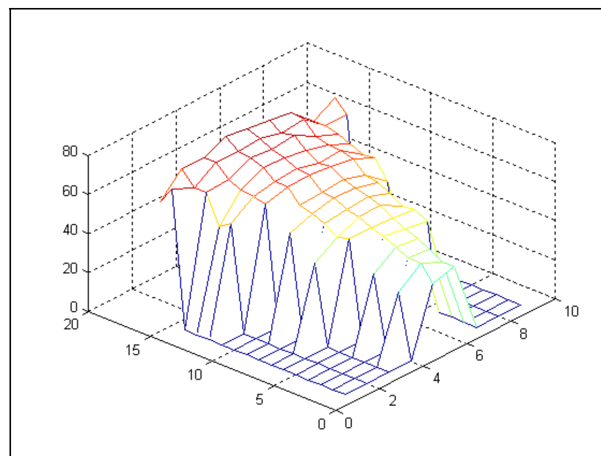

Figure 12. Surface mesh of one side

The final attempt at reconstruction was done with the Scatter3 function of Matlab, run for each non zero each element in the matrix. Since the test object is radially symmetrical in every direction, only one surface needed to be recreated. This surface was cloned and translated to be projected from all four sides. The result of this operation is the final output of the program, and clearly outlines the shape and volume of the test object.

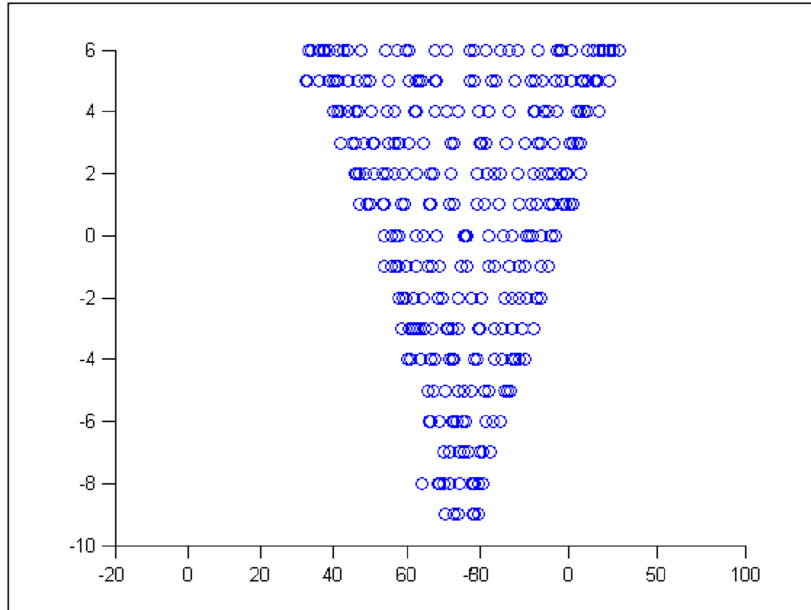

Figure 13. Final shape reconstruction

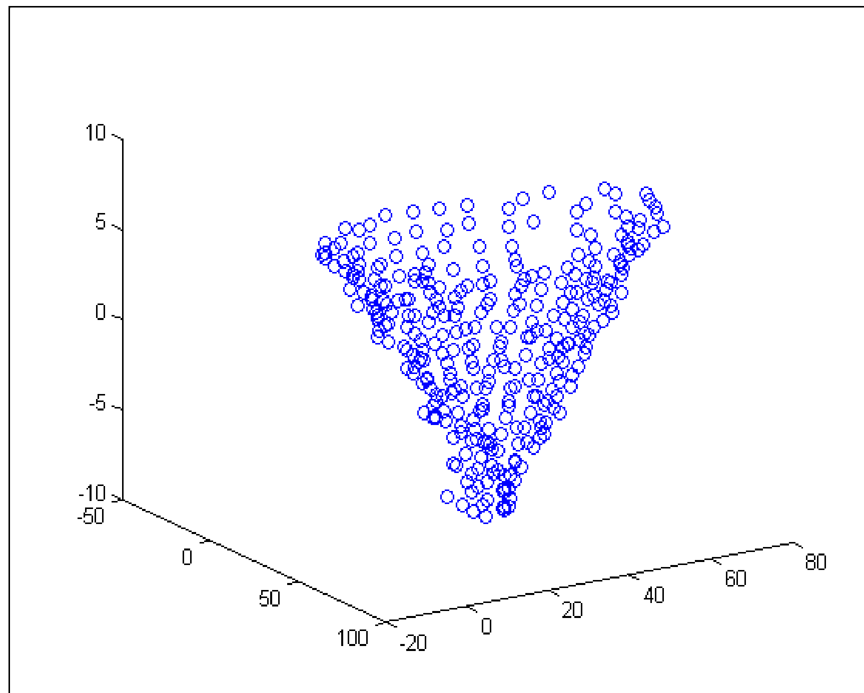

Figure 14. Final shape reconstruction

Conclusion

The results of the experiment described above, proved that a passive multi-camera system for 3D reconstruction is feasible. The system worked well to create an object of similar shape and proportions to its real world counter part. However, given more time and more money, a much more sophisticated system could be developed. Improvements would include, using multiple camera instead of a rotating platform, utilizing the linear regression model for smooth surface interpolation, connecting the point cloud with a meshing function, using reference images of the object to recover texture as well as shape, and testing on a non symmetrical object. Although these additional features would be nice, the current experiment yielded data consistent with the project goals, and is sufficient enough for proof of concept.

Appendix A: Builder Function

```
%Anil Rohatgi
% This program accepts multiple images of an object, and reconstructs the
% object in 3D

close all;
clear all;
load xxleft xx;
load yyleft yy;
load xxr;
load yyr;

left=imread('C:\data\Computer Vision\automated\autoleft.bmp');    %loads left image
right=imread('C:\data\Computer Vision\automated\autoright.bmp');  %loads right image

leftdouble=double(left)+1;      %Converts these images to doubles
rightdouble=double(right)+1;

%|________________________________________________________________________|
%|--------------------Getting Lines from left Image--------------------|
%|________________________________________________________________________|

%---------------------aquiring initial points--------------------
figure(1)
colormap(gray);
imagesc(leftdouble)        %obtaining initial conditions from mouse click
% [xx,yy]=ginput;
% xx=round(xx);             %make coordinates into ints
% yy=round(yy);

hold on
[leftlines,leftedge,leftedgemask]=lines(leftdouble,xx,yy);

% -----------------------plotting lines---------------------------------
for L=1:length(leftlines(:,1))
   plot(leftlines(L,1)*(1:length(leftdouble(1,:)))+leftlines(L,2),'r')
end
hold off

%|________________________________________________________________________|
%|---------------------Getting Lines from right image------------------|
%|________________________________________________________________________|

%---------------------aquiring initial points--------------------
figure(2)
```

```
colormap(gray);
imagesc(rightdouble)        %obtaining initial conditions from mouse click
% [xxr,yyr]=ginput;
% xxr=round(xxr);             %make coordinates into ints
% yyr=round(yyr);

hold on
[rightlines,rightedge,rightedgemask]=lines(rightdouble,xxr,yyr);

%------------------------plotting lines---------------------------------
for L=1:length(rightlines(:,1))
   plot(rightlines(L,1)*(1:length(rightdouble(1,:)))+rightlines(L,2),'r')
end

hold off
% ----------Take difference in lines from one image to the other------

diffmap = leftedge-rightedge;          %dispartiy map
totaledgemask=rightedgemask.*leftedgemask;
diffmapmask=diffmap.*totaledgemask;
[xnoz,ynoz]=find(diffmapmask);

ninescale=6;
xsub=-10;
figure(3)
hold on
% axis equal
% mesh(1:length(diffmapmask(1,:)),1:length(diffmapmask(:,1)),diffmapmask);
% mesh(1:length(opvec(1,:)),1:length(opvec(:,1)),opvec);
% mesh(xnoz,ynoz,diffmapmask);
for i=1:length(diffmapmask(:,1))
   for j=1:length(diffmapmask(1,:))
      if(diffmapmask(i,j)>0)
         scatter3(diffmapmask(i,j),j*ninescale,i+xsub);
         scatter3(j*ninescale,diffmapmask(i,j),i+xsub);
         scatter3(75-diffmapmask(i,j),j*ninescale,i+xsub)
         scatter3(j*ninescale,75-diffmapmask(i,j),i+xsub)
      end
   end
end
hold off

figure(4)
mesh(1:length(diffmapmask(1,:)),1:length(diffmapmask(:,1)),diffmapmask);
```

Appendix B: Correlation and Regression Function

```
%Anil Rohatgi
function [lines,edge,edgemask]= lines(picture,xx,yy)
%-----------------------------------------------------------------------
%This function takes an image of an object
%and finds the edges of the checker pattern at each input height, and then
%connects the correct poitns with lines. It returns the vector of line
%coefficients.
%-----------------------------------------------------------------------
leftdouble=picture;
%---------------------finding edge boundries-------------------
% hold on
scatter(xx,yy)            %plot clicked points
edge=zeros(2,2);      %initilize the edge array for 1st image
index=1;
for i=1:length(yy)                    %iterate over the heights
    for j=xx(i):length(leftdouble(1,:))-1    %iterate over the horizontal pixel values
(click:end-1)
        if(abs(leftdouble(yy(i),j)-leftdouble(yy(i),j+1))>220) %if difference between pixel
and next is >220
            rightedgeH(i,index)=j %then store it as edge
            edgemaskH(i,index)=1;
            %scatter(j,yy(i));           %plot the edge points
            index=index+1;                 %incriment store array
         end
     end
        index=1;                            %reset store array index
     for j2=xx(i):-1:2                 %iterate over the horizontal pixel values
(click:beginning+1)
        if(abs(leftdouble(yy(i),j2)-leftdouble(yy(i),j2+1))>220) %if difference between pixel
and next is >220
            rightedgeL(i,index)=j2; %then store it as edge
            edgemaskL(i,index)=1;
%             scatter(j2,yy(i));               %plot the edge points
            index=index+1;                   %incriment store array
         end
     end
   index=1;                           %reset store array
end


edge=[rightedgeL(:,length(rightedgeL(1,:)):-1:1),rightedgeH];          %combine edges
from left edge to right edge
edgemask=[edgemaskL(:,length(edgemaskL(1,:)):-1:1),edgemaskH];
%------------------------forming regression lines----------
```

```
xtemp=zeros(0,0);                    %reset temp vectors
ytemp=zeros(0,0);
tempindex=1;
for k2=1:length(edge(1,:))      %iterate over each edge index
   for ii=1:length(edge(:,1))  %iterate over height values
      if(edge(ii,k2)~=0)          %if the values isn't =0
         xtemp(tempindex)=edge(ii,k2);   %store its value in a temp variabl
         ytemp(tempindex)=yy(ii);        %store coorisponding Y value in temp variabl
         tempindex=tempindex+1;

      end
      scatter(xtemp,ytemp,'r')            %plot the non-zero edge points per height
      end
      xtemp
      ytemp
   line2=polyfit(xtemp,ytemp,1);           %form regession line
%   plot(line2(1)*(1:length(leftdouble(1,:)))+line2(2),'r')
   linearrayleft(k2,:)=line2;             %store regression coefficients in a matrix
   tempindex=1;                          %rest temp vector array index
   xtemp=zeros(0,0);                    %reset temp vectors
   ytemp=zeros(0,0);
end
edge
yy
lines=linearrayleft;

% hold off
```