COMP 4102 Final Project

3DReconstructor

Roy Xu 100999873

Jiangyuan Wu 100964177

Zhaohong Wan 101047896

Wayne Du 101013143

April 15, 2020

Carleton University

Abstract:

The 3DReconstructor is a 3-D image reconstruction tool that renders a three-dimensional model based on multiple 2-D images. Considering typical 3-D reconstruction system requires multiple cameras to proceed; classic cameras that mainly snap views from a single direction and cannot shape a naturalistic three-dimensional visualization. To facilitate 3D reconstruction, our project is implemented with several computer-vision methods that effectively accomplish the reconstruction process.

Introduction:

The core functionalities include object edge detection, 2-D image re-sizing, and 3-D mapping. Our project is intended to optimize the current 3-D reconstruction techniques, simplify the complex process by designing robust algorithm, and economize expenses as well as other resources on hardware equipment as conventionally multiple cameras are necessary.

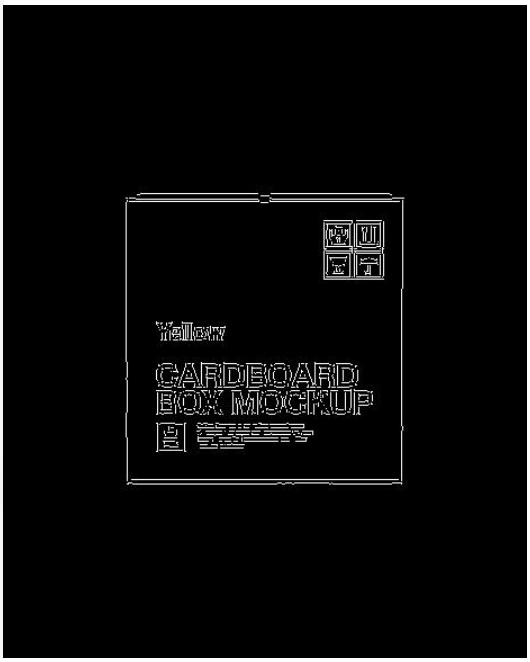
Background:

Our project is inspired by a tool called *Milkscanner* that allows the scanning of objects and creates a displacement map. The displacement map is produced by adding a brunch of cross-sections layer by layer. (Milkscanner, https://www.instructables.com/id/Milkscanner-V1.0/)

Milkscanner is an impressive invention since how mind-blowingly it produces displacement maps. However, it is debatable as the approach may require lengthy durations to gradually cover the surface and some objects are not appropriate to be experimented with milk such as objects in non-solid forms or solid forms but complicated shapes. Our design is to generate displacement maps achieved by three-views-drawing. Our 3-D Reconstructor will use three-views-drawing to determine the position of a point, which is on the surface of our object, in three dimensions space.

Approach:

Edge detection:



Since we want to gather the information of the object from its image, the good way we can use is to using edge detection to help our program understand the size and the detail of the object. The purpose of edge detection is to identify points in a digital image that has a significant change of the brightness. So we think edge detection can be really helpful while we are collecting the shape of the object. First, we convert the object's image into grayscale to guarantee that we can access the pixel value only by its row and column coordinates. It can help us to ignore the BGR value in the coloured image. Then we used the Gaussian filter to smooth the image so we are able to remove the noise.

Gaussian filter formula:

$$H_{ij} = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{(i-(k+1))^2 + (j-(k+1))^2}{2\sigma^2}\right); 1 \leq i, j \leq (2k+1)$$

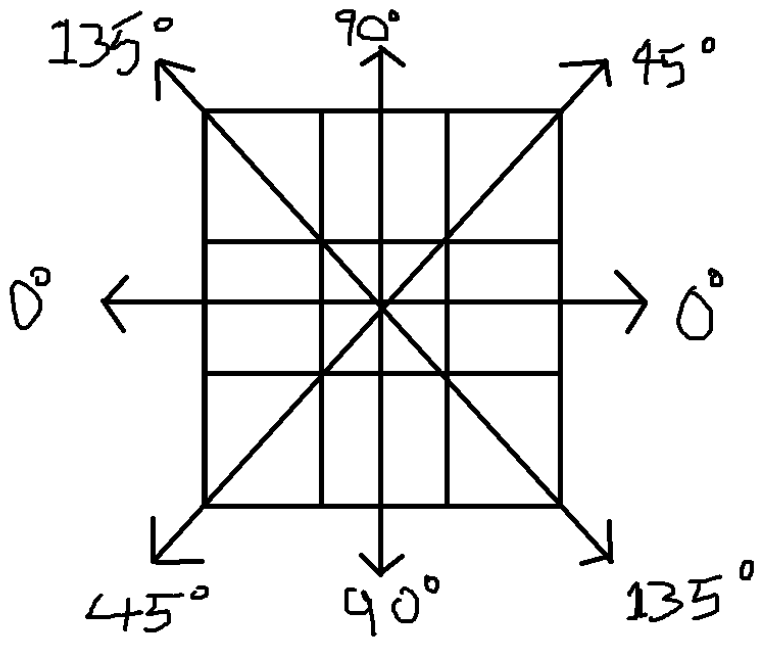
Then, we used the Sobel operator to find the intensity gradient of the object image. Based on that, we can get the value for the first derivative in the horizontal direction and the vertical direction. From that, we can determine the edge gradient and the direction.

$$\mathbf{G} = \sqrt{\mathbf{G}_x^{\,2} + \mathbf{G}_y^{\,2}} \qquad \mathbf{\Theta} = \operatorname{atan2}(\mathbf{G}_y, \mathbf{G}_x)$$

Where:

$$\mathbf{G}_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * \mathbf{A} \quad \text{and} \quad \mathbf{G}_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * \mathbf{A}$$

Then we need to use the non-maximum suppression to thinner the edge which we had detected from the image. Since we had used the Sobel operator for the gradient calculation, we can use the non-maximum suppression to suppress all the gradient value except the local maxima, that indicate the location with the sharpest change of intensity value. This technique usually worked as: Compare the edge strength of the current pixel with the edge strength of the pixel in the positive and negative gradient direction. And the current pixel value will be preserved, if the edge strength of the current pixel is the largest when compared to the other pixel which at the same direction. If the pixel value is not the largest, then the value will be suppressed.



Then we using the threshold function to transform the weak pixel into a strong pixel, only if the pixel value has passed the assigned threshold value. The purpose of this process is to separate the strong pixels from the other weak pixels. And meanwhile, this process provides a clearer picture to let our program gathering data from.

Threshold algorithm:

```
//M(x,y) represents a pixel of the picture
If M(x,y) > threshold value:
        M(x,y) = strong(usually be 255)
Else:
        M(x,y) = 0
```

Resizing:

To make sure when we are mapping front, left and top view images, every pixel on one image can have the specific third axis value to make a 3D model. Front, left and top view images must have their relative size

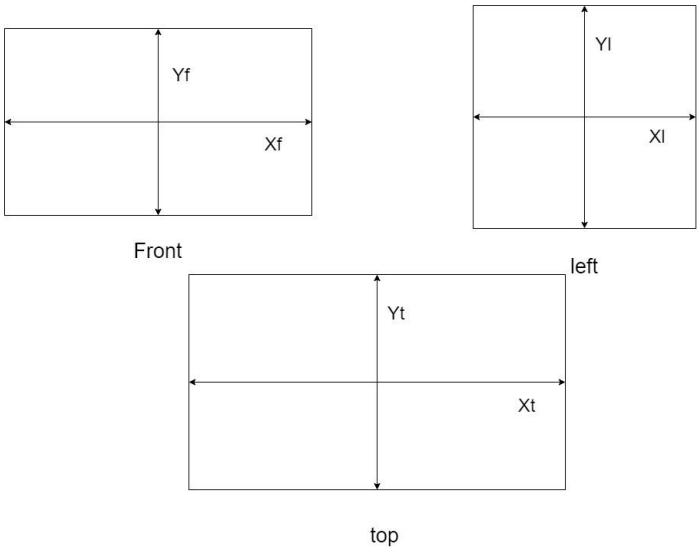


Figure 1: This is showing the three view images for a cuboid

Considering the three view images which is showing on Figure 1, our resized images must meet following requirements:

$$1: Xf = Xt$$

$$2: Yf = Yl$$

$$3: Yt = Xl$$

Therefore, when we want to find the third axis value of a point on one view image, for example, one point is on front view image, we can easily find the Z value for this point based on left or top view image. Because after resizing, front, left and top images are in the same scale.

To do the resizing, first thing is defining the length of the object in x-axis and y-

axis on front, left and top view images. For each view image, resize function search the image from top to bottom first, if function finds the y value that defines the object, then it will record this y-axis value. Then, resize function will do the same thing from bottom to top, this time function will find another y-axis value and the difference between these to y value this the length of the object in y-axis on this image. For the length of object in x-axis, resize function will search from left to right and right to left and record two x values. Finally, it will calculate the difference between these two values.

Next step is to determine which image is the basic image that we will resize two other images based on its scale. Because we do not want to lose pixel after resizing images, so we always choose the image with the smallest scale in three view images. According to the length of the object in x-axis and y-axis on front, left and top view images, we can find which image has the smallest scale. For example:

$$when\ \frac{length\ of\ object\ in\ x-axis\ on\ front\ image}{length\ of\ object\ in\ x-axis\ on\ top\ image} < 1$$

$$and\ \frac{length\ of\ object\ in\ x-axis\ on\ front\ image}{length\ of\ object\ in\ x-axis\ on\ left\ image} < 1$$

For this condition, front view image has the smallest scale.

Or

$$when\ \frac{length\ of\ object\ in\ x-axis\ on\ front\ image}{length\ of\ object\ in\ x-axis\ on\ top\ image} > 1$$

$$and\ \frac{length\ of\ object\ in\ x-axis\ on\ front\ image}{length\ of\ object\ in\ x-axis\ on\ left\ image} < 1$$

For this condition, top view image has the smallest scale.

After deciding which two images will be resized, our program will calculate and record two ratios between basic image and resized images. When we are using the scale of front image:

$$ratio1 = front\ y\ length/left\ y\ length$$

$$ratio2 = front\ x\ length/top\ x\ length$$

For the scale of top image:

$$ratio1 = top\ y\ length/left\ x\ length$$

$$ratio2 = top\ x\ length/front\ x\ length$$

For the scale of left image:

$$ratio1 = left\ x\ length/top\ y\ length$$

$$ratio2 = left\ y\ length/front\ y\ length$$

With these two ratios, we shrink the whole image with larger scale by OpenCV library function—cv2.resize(). Cv2.resize() needs to input parameter, first is the image we want to resize, second is a pair of data with the length of x-axis value and the length of y-axis value which determines the size of image after resizing. We have the two original images, which need to resize, and their two lengths, the length of two axis value after resizing can be calculated by: $length * ratio$.

Final step for resizing is filling up the shrunken images. The objects on two resized image and the object on basic image are under identical scale, however, they have different image size. To facilitate the mapping step, resize function will fill up the border with OpenCV library function—cv2.copyMakeBorder, in order to make shrunken images have the same image size as the basic image. Cv2.copyMakeBorder need 6 input parameters, first one is the image need to fill up, next four parameters require the depth of border for top, bottom, left and right side. Last parameter we use cv2.BORDER_REPLICATE to indicate our new border will have the same pixel as the original border pixel on the shrunken image.

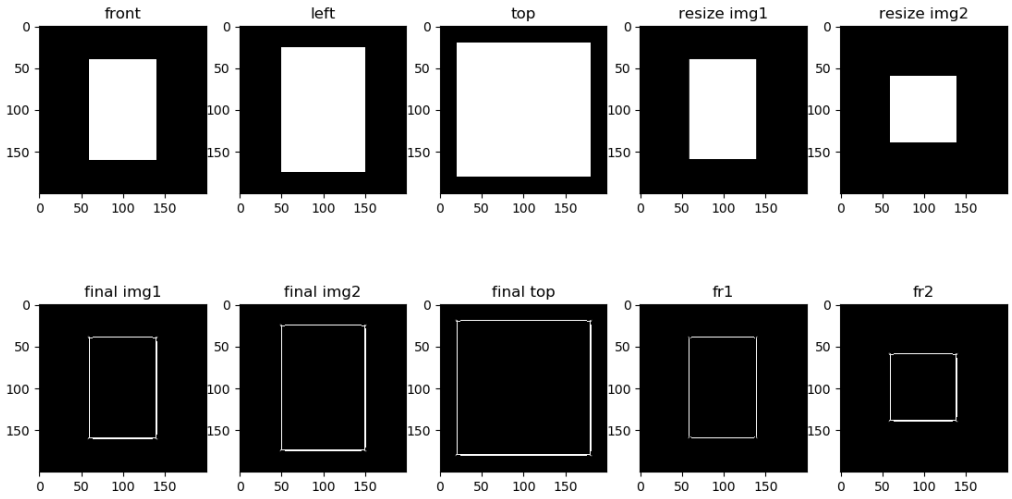


Figure 2: This graph is showing the original images of front, left and top view images

and the resized images of left and top images. Front image has the smallest scale; therefore, two resized images are left and top images.

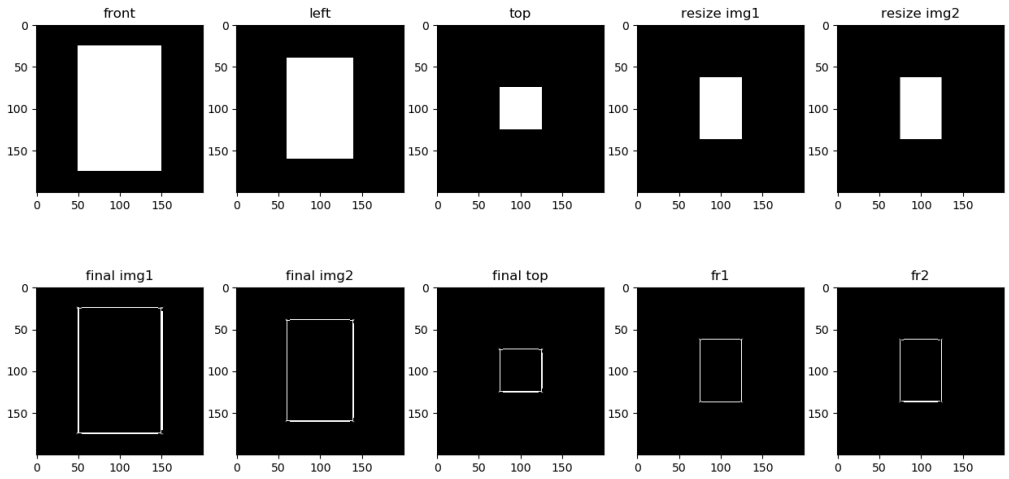


Figure 3: This graph is showing the original images of front, left and top view images and the resized images of front and left images. Top image has the smallest scale; therefore, two resized images are left and front images.

Point Mapping:

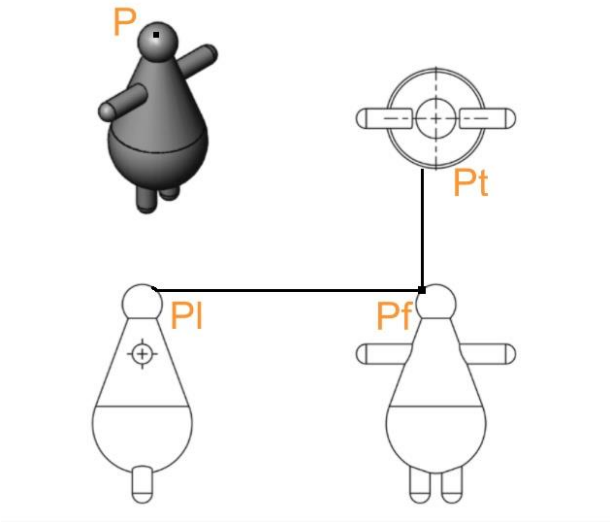


Figure 4: Three views of point P (Pf, Pl, Pt)

Any 3D object could be treated as a collection of points inside 3D space, and all point in the collection has three dimensional values corresponding to each axis (denoted as x, y, z). As shown in the Figure 4, the x and z value of an arbitrary point P could be gotten from the front view of the object. The y value could be gotten from left as Pl or from top as Pt. In order to get the true value of y, the y value needs to be chosen as the point that has shorter distance from the center of that view. For instance, the point Pf could be mapping to Pl or Pt. As shown in the 3D view on the left top of Figure 4, the true value of y should correspond to the Pl instead of Pt, hence here after we compare the vertical distance from Pt to its vertical center with the horizontal

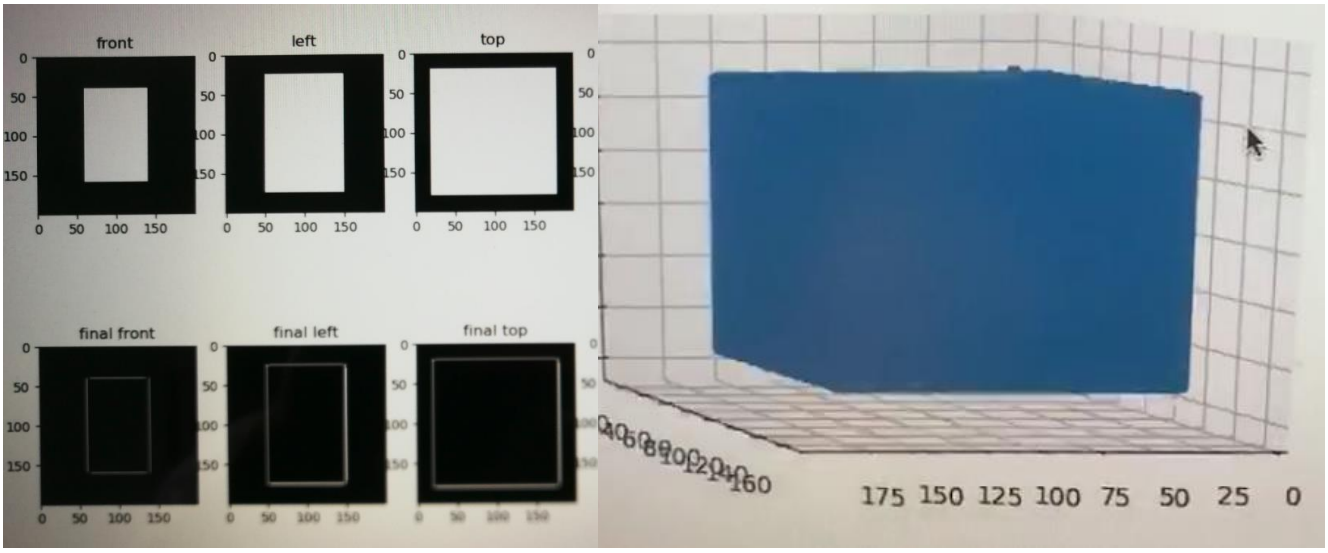


Figure 5: The 2D 3-Views and Reconstructed 3D Object

distance between Pl to its horizontal center and then choose the shorter one (the Pl). So the y value of P will be assigned to the horizontal distance between Pl to its horizontal center. After doing the same procedure for all the points on the object, we got the reconstructed 3D object from 2D images and plotted by peplos shown in Figure 5.

The Algorithm

```
//map front and back surface
For each row in the Front-View Do:
        For each column in the row Do:
              //append the point with coordinate (x, y, z) to object
              object.append(column, Min(y from left, y from top), row)

//map left and right surface
For each row in the Left-View Do:
        For column element in the row Do:
              //append the point with coordinate (x, y, z) to object
              object.append(Min(x from top, x from front), column, row)

//map top and bottom surface
For each row in the Top-View Do:
        For column element in the row Do:
              //append the point with coordinate (x, y, z) to object
              object.append(column, row, Min(z from left, z from front))
```

Extract:

Example picture:

The purpose of the extract() function is to crop the object that the input image has, then we can use the cropped image to build the 3D model. In this function, we used three built-in functions that already exist in the OpenCV library: threshold(), findContours() and boundingRect(). First, we used the threshold() function to set the thresh value to 100, and the pixel value will be assigned to the maxval(which is 255) if the pixel value is higher than the thresh value. By using this way, we can easily separate the object from the input image. Then, we used the findContours() function to detect the contour of the object. As we knew, the hierarchy exists in images, it represents the relationship of the shapes that inside another shape. But we do not need to worry about that, so we using the RETR_LIST as the type of the findContours() function to simply retrieves all the contours, but doesn't create any parent-child relationship, so all the pixels in this image all belong to the same hierarchy level. Since we wanted to separate the object from the image, we used the boundingRect() function to build the smallest square which can exactly cover the object that we want to separate. This function will return four things: the square's x value of the top left corner, the square's y value of the top left corner, the width of the square and the height of the square. By getting that

information, we can easily do the separation.

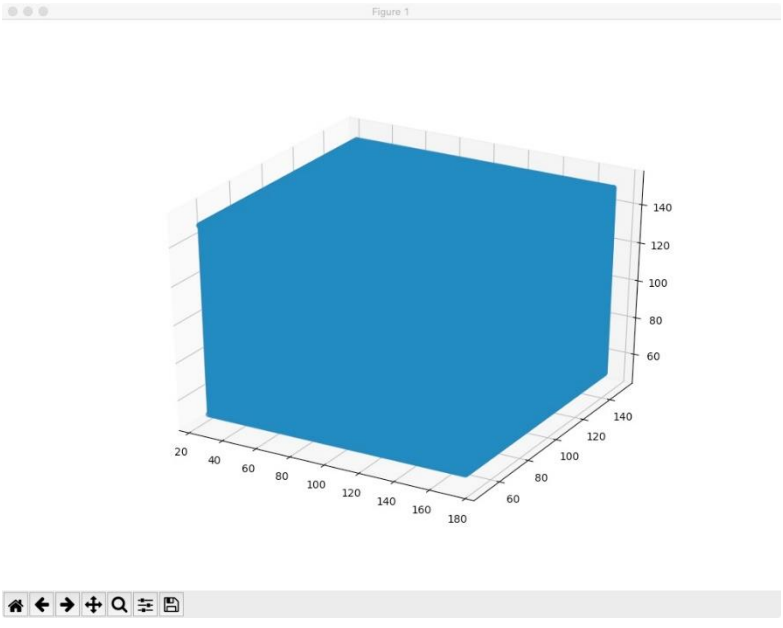
Algorithm for threshold:

```
Add-in matrix M;
//M(x, y) = coordinate pixel value
If M(x, y) > threshold value(assigned or not):
        Then matrix M is 0
Else:
        Matrix M is maxval(usually assigned to 255)
```

Result:

Sample Input:

3-D Reconstruction:



List of Work:

Equal work was performed by all project members.

GitHub Page:

https://github.com/R2333333/3DReconstructor

References:

1. Adrian Rosebrock, “Non-Maximum Suppression for Object Detection in Python” https://www.pyimagesearch.com/2014/11/17/non-maximum-suppression-object-detection-python/ , (November 17, 2014)

2. Rosa Azami, COMP 4102 Course Materials

3. Adrian Rosebrock, “Building a Pokedex in Python: Finding the Game Boy Screen (Step 4 of 6)”. https://www.pyimagesearch.com/2014/04/21/building-pokedex-python-finding-game-boy-screen-step-4-6/ ,(April 21, 2014)

4. Doxygen. (2020, April 18). resize(). Retrieved from: https://docs.opencv.org/trunk/da/d54/group__imgproc__transform.html#ga47a974309e9102f5f08231edc7e7529d

5. Doxygen. (2020, April 18). copyMakeBorder(). Retrieve from: https://docs.opencv.org/trunk/d2/de8/group__core__array.html#ga2ac1049c2c3dd25c2b41bffe17658a36