Project Progress Update for 3D Reconstruction Project

Ge Jiaxin, 58153795

The final project is about reconstruction of scanned images by utilizing techniques that we learned in class. Below is the outline of steps/milestones that needed to finish in the project:

1. Reorganize code from assignments and sort out in camutils.py files (Completed)
2. Compute intrinsic and extrinsic parameters for left and right camera (Completed)
3. Modify reconstruct function to collect colors and foreground object (Completed)
4. Reduce noise before meshing (In progress)
5. Smooth mesh by computing the average location of the points neighbor (Not Started)
6. Save each meshes as .ply file and load in Meshlab (Not Started)
7. Apply Poisson surface reconstruction to .ply files (Not Started)
8. Render final model in Maya/Blender and prepare for the final report (Not Started)

I refine functions from previous four assignments and replace original functions provided by the professors with my own functions, including makerotation(), class Camera, triangulate(), residuals() and calibratePose(). I replace variables to become more readable and write notes on each step that I take (which will be useful when I prepare for my final report and explain each step I take). Figure 1 shows part of the code from triangulate() function.

```
---------
--------
pts3 : 2D numpy.array (dtype=float)
    (3,N) array containing 3D coordinates of the points in global coordinates

"""
n = pts2L.shape[1]
# calculate qL and qR by using formula: (p-offset)/focal length
qL = np.append((pts2L-camL.c)/camL.f, np.ones(n).reshape(1, n), axis = 0)
qR = np.append((pts2R-camR.c)/camR.f, np.ones(n).reshape(1, n), axis = 0)
pts3_L = np.ones((3, n))
pts3_R = np.ones((3, n))

for index in range(n):

    # calculate A and b in the least square equation Au = b
    A1 = np.matmul(camL.R, qL[:,index].reshape(3,1))
    A2 = np.matmul(-camR.R, qR[:,index].reshape(3,1))
    A = np.append(A1, A2, axis = 1)
    b = camR.t-camL.t

    # using np.linalg.lstsq to calculate u in two dimension (z point)
    ZL, ZR = np.linalg.lstsq(A, b)[0]
    PL = ZL*qL[:,index]
    PR = ZR*qR[:,index]

    # change from camera projection to world coordinate and add z position
    pts3_L[:, index] = (np.matmul(camL.R, PL).reshape(3, 1)+camL.t).reshape(1, 3)
    pts3_R[:, index] = (np.matmul(camR.R, PR).reshape(3, 1)+camR.t).reshape(1, 3)

pts3 = 1/2 * (pts3_L+pts3_R)
return pts3
```

Figure 1: Readable variables and notes for each steps

Then I computed extrinsic and intrinsic parameters for the two cameras, given the calibrated images of the scanned teapot. I load calibration images from the calib_jpg_u file that was given, and apply to pickle.py to create calibration.pickle, which gave me the intrinsic parameters for my left and right camera. Then use cv2.findChessboardCorners from opencv to find the coordinates of the chessboard corners and np.meshgrid() to find the coordinates of 3D points. I applied these parameters to calibratePose() and used scipy.optimize.leastsq() to compute the estimated 3D projected points and the camera extrinsic parameters. Figure 2 is the estimated intrinsic and extrinsic parameters of both left and right camera.

```
camL = calibratePose(pts3, pts2L, camL, np.array([0,0,0,1,1,-1]))
camR = calibratePose(pts3, pts2R, camR, np.array([0,0,0,0,0,-1]))
```

```
: print (camL)
  print (camR)
```

```
Camera :
 f=1404.6009650962417
 c=[[962.16736421 590.91595766]]
 R=[[ 0.03843674  0.98947412  0.13951197]
 [ 0.9773577  -0.00815434 -0.21143658]
 [-0.20807339  0.14448003 -0.96738358]]
 t = [[ 6.86588616 19.52347083 47.34419138]]
Camera :
 f=1404.6009650962417
 c=[[962.16736421 590.91595766]]
 R=[[-0.00259871  0.99096865  0.13406858]
 [ 0.99277875 -0.01352251  0.11919517]
 [ 0.11993162  0.1334102  -0.98377748]]
 t = [[ 7.50010466  7.20926449 47.76495322]]
```

Figure 2: Extrinsic and intrinsic parameters of two cameras

After that, I modify the reconstruct function from assignment 4. Given the foreground and background images, I created a mask which computed the difference between them and determined whether the pixel is greater than the given threshold. I applied the results and multiplied by the decoded masks (which used to decode 10 bit grey code to a decimal value and return the mask). I also saved the color from one of the foreground images, which will be used to color the mesh in the future. Though I have finished this step, I still want to use both images to refine my result, which is a difficulty that I want to encounter. Figure 3 shows part of my modified code.

```
# Read foreground and background for the scanned objects
leftB = plt.imread(prefix+ "/color_C0_00.png")
leftF = plt.imread(prefix + "/color_C1_01.png")
rightB = plt.imread(prefix + "/color_C0_00.png")
rightF = plt.imread(prefix + "/color_C1_01.png")

# Threshold the foreground and background
objMaskL = np.array(np.abs(leftF-leftB)>threshold)
objMaskR = np.array(np.abs(rightF-rightB)>threshold)

# Construct the combined 20 bit code C = H + 1024*V and mask for each view
CL = (HL + 1024 * VL) * HLmask * VLmask * objMaskL
CR = (HR + 1024 * VR) * HRmask * VRmask * objMaskR

# Find the indices of pixels in the left and right code image that
# have matching codes. If there are multiple matches, just
# choose one arbitrarily.
Lmatching = np.intersect1d(CL, CR, return_indices=True)[1]
Rmatching = np.intersect1d(CR, CL, return_indices=True)[1]

# Let CL and CR be the flattened arrays of codes for the left and right view
# Suppose you have computed arrays of indices matchL and matchR so that
# CL[matchL[i]] == CR[matchR[i]] for all i.  The code below gives one approach
# to generating the corresponding pixel coordinates for the matched pixels.
CL = CL.flatten()
CR = CR.flatten()
h = HL.shape[0]
w = HL.shape[1]

xx,yy = np.meshgrid(range(w),range(h))
xx = np.reshape(xx,(-1,1))
yy = np.reshape(yy,(-1,1))
pts2R = np.concatenate((xx[Rmatching].T,yy[Rmatching].T),axis=0)
pts2L = np.concatenate((xx[Lmatching].T,yy[Lmatching].T),axis=0)

# Now triangulate the points
pts3=triangulate(pts2L,camL,pts2R,camR)

#Record the color from one of the images
color_mask = leftF[pts2L[1,:], pts2L[0,:],:].T
assert color_mask.shape = np.ones((h,w))

return pts2L,pts2R,pts3D,color_mask
```

Figure 3: Code that modified reconstruct function

Currently, I'm in progress dealing with the problem to reduce noise from the mesh() function. I set up a range as limitation to x, y, and z coordinate for bounding box pruning. I run Delaunay to keep good points. I'm still struggling with how to do triangle pruning, because I haven't finished it in my assignment 4. The noise reduction for triangle pruning does not perform very well.

I haven't started from steps 4 to steps 8. For smoothing, I need to figure out how to compute and replace the average distances for each 3D point with their neighbours. Since I choose to load the scans of points into Meshlab instead of using SVD, I wouldn't need to worry about the algorithm of SVD, but I need to figure out how to apply to Meshlab. Then I need to use the urls provided in resources pages and apply Poisson surface reconstruction to my mesh, which can be another difficulty. And finally, I need to apply the mesh into Maya or Blender to view it in a render view. I need to figure out how to apply the model into these software.