CS 532: Homework Assignment 3

Code:

% Silhouette Images

s0 = imread('silh\_cam00\_00023\_0000008550.pbm');

s1 = imread('silh\_cam01\_00023\_0000008550.pbm');

s2 = imread('silh\_cam02\_00023\_0000008550.pbm');

s3 = imread('silh\_cam03\_00023\_0000008550.pbm');

s4 = imread('silh\_cam04\_00023\_0000008550.pbm');

s5 = imread('silh\_cam05\_00023\_0000008550.pbm');

s6 = imread('silh\_cam06\_00023\_0000008550.pbm');

s7 = imread('silh\_cam07\_00023\_0000008550.pbm');

% Concatenate silhouette images into a 3D array

s = zeros(582, 780, 8);

s(:,:,1) = s0; s(:,:,2) = s1; s(:,:,3) = s2; s(:,:,4) = s3;

s(:,:,5) = s4; s(:,:,6) = s5; s(:,:,7) = s6; s(:,:,8) = s7;

% Color Images

i0 = imread('cam00\_00023\_0000008550.png');

i1 = imread('cam01\_00023\_0000008550.png');

i2 = imread('cam02\_00023\_0000008550.png');

i3 = imread('cam03\_00023\_0000008550.png');

i4 = imread('cam04\_00023\_0000008550.png');

i5 = imread('cam05\_00023\_0000008550.png');

i6 = imread('cam06\_00023\_0000008550.png');

i7 = imread('cam07\_00023\_0000008550.png');

% Concatenate color images into a 4D array

im = zeros(582, 780, 3, 8);

im(:,:,:,1) = i0; im(:,:,:,2) = i1; im(:,:,:,3) = i2; im(:,:,:,4) = i3;

im(:,:,:,5) = i4; im(:,:,:,6) = i5; im(:,:,:,7) = i6; im(:,:,:,8) = i7;

% Camera Projection Matrices

pm0 = [776.649963 -298.408539 -32.048386 993.1581875;132.852554 120.885834 -759.210876 1982.174000;0.744869 0.662592 -0.078377 4.629312012];

pm1 = [431.503540 586.251892 -137.094040 1982.053375;23.799522 1.964373 -657.832764 1725.253500;-0.321776 0.869462 -0.374826 5.538025391];

pm2 = [-153.607925 722.067139 -127.204468 2182.4950;141.564346 74.195686 -637.070984 1551.185125;-0.769772 0.354474 -0.530847 4.737782227];

pm3 = [-823.909119 55.557896 -82.577644 2498.20825;-31.429972 42.725830 -777.534546 2083.363250;-0.484634 -0.807611 -0.335998 4.934550781];

pm4 = [-715.434998 -351.073730 -147.460815 1978.534875;29.429260 -2.156084 -779.121704 2028.892750;0.030776 -0.941587 -0.335361 4.141203125];

pm5 = [-417.221649 -700.318726 -27.361042 1599.565000;111.925537 -169.101776 -752.020142 1982.983750;0.542421 -0.837170 -0.070180 3.929336426];

pm6 = [94.934860 -668.213623 -331.895508 769.8633125;-549.403137 -58.174614 -342.555359 1286.971000;0.196630 -0.136065 -0.970991 3.574729736];

pm7 = [452.159027 -658.943909 -279.703522 883.495000;-262.442566 1.231108 -751.532349 1884.149625;0.776201 0.215114 -0.592653 4.235517090];

% Concatenate projection matrices into a 3D array

pm = zeros(3,4,8);

pm(:,:,1) = pm0; pm(:,:,2) = pm1; pm(:,:,3) = pm2; pm(:,:,4) = pm3;

pm(:,:,5) = pm4; pm(:,:,6) = pm5; pm(:,:,7) = pm6; pm(:,:,8) = pm7;

%Step1

x\_grid = 5; % Number of voxels along the x-axis

y\_grid = 6; % Number of voxels along the y-axis

z\_grid = 2.5; % Number of voxels along the z-axis

vol = x\_grid \* y\_grid \* z\_grid; % Total volume of the voxel grid

no\_voxs = 10000000; % Desired number of voxels

vox\_s = nthroot(vol/no\_voxs,3); % Size of each voxel

true\_no\_voxs = 0; % Actual number of voxels

total\_no\_voxs = 0; % Total number of voxels processed

vox\_mat = []; % Voxel matrix

sur\_vox\_mat = []; % Surface voxel matrix

color\_mat = []; % Color matrix

fl = 0; % Flag variable

prev\_vec = []; % Previous vector

voxel\_labels = zeros(round(x\_grid/vox\_s), round(y\_grid/vox\_s), round(z\_grid/vox\_s)); % Matrix to store voxel labels

for x\_idx = 1:size(voxel\_labels, 1)

for y\_idx = 1:size(voxel\_labels, 2)

for z\_idx = 1:size(voxel\_labels, 3)

x = (x\_idx - 1) \* vox\_s - x\_grid/2;

y = (y\_idx - 1) \* vox\_s - y\_grid/2;

z = (z\_idx - 1) \* vox\_s;

dec\_v = [0 0 0 0 0 0 0 0];

total\_no\_voxs = total\_no\_voxs + 1;

w\_col = [x y z 1.0].';

for i = 1:8

uv\_cor = pm(:,:,i)\*w\_col;

uv\_cor = round(uv\_cor/uv\_cor(3));

if (1<=uv\_cor(1)) && (uv\_cor(1)<=780) && (1<=uv\_cor(2)) && (uv\_cor(2)<=582)

dec\_v(i) = s(uv\_cor(2),uv\_cor(1),i);

end

end

if all(dec\_v) == 1

true\_no\_voxs = true\_no\_voxs + 1;

vox\_mat = [vox\_mat;[x y z]];

r = im(uv\_cor(2), uv\_cor(1), 1, 8);

g = im(uv\_cor(2), uv\_cor(1), 2, 8);

b = im(uv\_cor(2), uv\_cor(1), 3, 8);

color\_mat = [color\_mat;[r g b]];

% Step3 The code below is for detecting surface voxels

if fl == 0

sur\_vox\_mat = [sur\_vox\_mat;[x y z]];

fl = fl+1;

prev\_vec = [x y z];

continue;

end

if (prev\_vec(1)==x) && (prev\_vec(2)==y)

fl = fl+1;

else

if fl > 1

sur\_vox\_mat = [sur\_vox\_mat;prev\_vec;[x y z]];

f1=1;

else

sur\_vox\_mat = [sur\_vox\_mat;[x y z]];

f1=1;

end

end

if fl > 0

prev\_vec = [x y z];

end

end

% Step 2: Determine the voxels forming the visual hull

% Project the 3D center of the voxel onto all images and check if it lies inside all silhouettes

occluded = false; % Flag to check if voxel is occluded by any silhouette

for i = 1:8

uv\_cor = pm(:,:,i) \* [x y z 1]';

uv\_cor = round(uv\_cor / uv\_cor(3));

% Check if the projection lies inside the silhouette

if (1 <= uv\_cor(1)) && (uv\_cor(1) <= 780) && (1 <= uv\_cor(2)) && (uv\_cor(2) <= 582)

if s(uv\_cor(2), uv\_cor(1), i) == 0

occluded = true; % Voxel is occluded by this silhouette

break;

end

else

occluded = true; % Projection lies outside image bounds, consider occluded

break;

end

end

% Label the voxel as OCCUPIED if not occluded by any silhouette

if ~occluded

voxel\_labels(x\_idx, y\_idx, z\_idx) = 1;

end

end

end

end

% Step 3: Identify surface voxels

% Surface voxels are OCCUPIED voxels having at least one neighboring voxel labeled as EMPTY.

sur\_voxel\_labels = zeros(size(voxel\_labels)); % Matrix to store surface voxel labels

for x\_idx = 2:size(voxel\_labels, 1)-1

for y\_idx = 2:size(voxel\_labels, 2)-1

for z\_idx = 2:size(voxel\_labels, 3)-1

if voxel\_labels(x\_idx, y\_idx, z\_idx) == 1

% Check if the voxel has at least one neighboring voxel labeled as EMPTY

if any(voxel\_labels(x\_idx-1:x\_idx+1, y\_idx-1:y\_idx+1, z\_idx-1:z\_idx+1) == 0)

sur\_voxel\_labels(x\_idx, y\_idx, z\_idx) = 1; % Mark as surface voxel

end

end

end

end

end

% Step 4: Determine the set of 3D points to include in the output

% From the surface voxels obtained in Step 3, add the 3D point of the center of each face that neighbors with an EMPTY VOXEL

%output\_points = []; % Matrix to store output 3D points

%for x\_idx = 2:size(sur\_voxel\_labels, 1)-1

%for y\_idx = 2:size(sur\_voxel\_labels, 2)-1

%for z\_idx = 2:size(sur\_voxel\_labels, 3)-1

% Check if the voxel is a surface voxel and neighbors with an EMPTY voxel

%if sur\_voxel\_labels(x\_idx, y\_idx, z\_idx) == 1 && any(any(any(voxel\_labels(x\_idx-1:x\_idx+1, y\_idx-1:y\_idx+1, z\_idx-1:z\_idx+1) == 0)))

% Calculate the center of the face

%x\_center = (x\_idx - 1) \* vox\_s - x\_grid/2 + vox\_s/2;

%y\_center = (y\_idx - 1) \* vox\_s - y\_grid/2 + vox\_s/2;

%z\_center = (z\_idx - 1) \* vox\_s + vox\_s/2;

%output\_points = [output\_points; [x\_center, y\_center, z\_center]];

%end

%end

%end

%end

% Step 4: Determine the set of 3D points to include in the output

% From the surface voxels obtained in Step 3, add the 3D point on the surface of the voxel adjacent to an EMPTY VOXEL

output\_points = []; % Matrix to store output 3D points

for x\_idx = 2:size(sur\_voxel\_labels, 1)-1

for y\_idx = 2:size(sur\_voxel\_labels, 2)-1

for z\_idx = 2:size(sur\_voxel\_labels, 3)-1

% Check if the voxel is a surface voxel and neighbors with an EMPTY voxel

if sur\_voxel\_labels(x\_idx, y\_idx, z\_idx) == 1 && any(any(any(voxel\_labels(x\_idx-1:x\_idx+1, y\_idx-1:y\_idx+1, z\_idx-1:z\_idx+1) == 0)))

% Calculate the coordinates of the surface points on the voxel

x\_left = (x\_idx - 1) \* vox\_s - x\_grid/2;

x\_right = (x\_idx - 1) \* vox\_s - x\_grid/2 + vox\_s;

y\_front = (y\_idx - 1) \* vox\_s - y\_grid/2;

y\_back = (y\_idx - 1) \* vox\_s - y\_grid/2 + vox\_s;

z\_bottom = (z\_idx - 1) \* vox\_s;

z\_top = (z\_idx - 1) \* vox\_s + vox\_s;

% Add points on the surface in the direction of neighboring empty voxels

if voxel\_labels(x\_idx-1, y\_idx, z\_idx) == 0

output\_points = [output\_points; [x\_left, (y\_front + y\_back) / 2, (z\_bottom + z\_top) / 2]];

end

if voxel\_labels(x\_idx+1, y\_idx, z\_idx) == 0

output\_points = [output\_points; [x\_right, (y\_front + y\_back) / 2, (z\_bottom + z\_top) / 2]];

end

if voxel\_labels(x\_idx, y\_idx-1, z\_idx) == 0

output\_points = [output\_points; [(x\_left + x\_right) / 2, y\_front, (z\_bottom + z\_top) / 2]];

end

if voxel\_labels(x\_idx, y\_idx+1, z\_idx) == 0

output\_points = [output\_points; [(x\_left + x\_right) / 2, y\_back, (z\_bottom + z\_top) / 2]];

end

if voxel\_labels(x\_idx, y\_idx, z\_idx-1) == 0

output\_points = [output\_points; [(x\_left + x\_right) / 2, (y\_front + y\_back) / 2, z\_bottom]];

end

if voxel\_labels(x\_idx, y\_idx, z\_idx+1) == 0

output\_points = [output\_points; [(x\_left + x\_right) / 2, (y\_front + y\_back) / 2, z\_top]];

end

end

end

end

end

% Calculate the total number of voxels in visual hull

total\_in\_visual\_hull = sum(voxel\_labels(:));

% Calculate the relative portion of all voxels in the visual hull

relative\_portion = total\_in\_visual\_hull / total\_no\_voxs;

% Display results

disp(['Total number of voxels in the visual hull: ', num2str(total\_in\_visual\_hull)]);

disp(['Relative portion of all voxels in the visual hull: ', num2str(relative\_portion)]);

% Calculate the total number of surface voxels

total\_surface\_voxels = size(sur\_vox\_mat, 1);

% Calculate the relative portion of all voxels that are surface voxels

relative\_surface\_portion = total\_surface\_voxels / total\_no\_voxs;

% Display results

disp(['Total number of surface voxels: ', num2str(total\_surface\_voxels)]);

disp(['Relative portion of all voxels that are surface voxels: ', num2str(relative\_surface\_portion)]);

% Display the number of 3D points included in the model

disp(['Number of 3D points included in the model: ', num2str(size(output\_points, 1))]);

% Compare the number of 3D points included in the model with the number of surface voxels

disp(['Number of surface voxels: ', num2str(total\_surface\_voxels)]);

% Discuss why the numbers may or may not be the same

%Step7 and Step8

vox\_mat\_ptc = pointCloud(vox\_mat);

vox\_mat\_ptc.Color = uint8(color\_mat);

pcwrite(vox\_mat\_ptc,'dancer\_full\_colored\_2','PLYFormat','ascii');

full\_mod = pcread('dancer\_full\_colored\_2.ply');

pcshow(full\_mod);

pcwrite(full\_mod,"dancer\_full\_colored\_2.ply");

%The code below is for creating a PLY file for the point cloud

%created by using only surface voxels.

% Assuming sur\_vox\_mat has n rows (points)

[n, ~] = size(sur\_vox\_mat);

% If color\_mat has more rows than sur\_vox\_mat, trim it

if size(color\_mat, 1) > n

color\_mat = color\_mat(1:n, :);

end

% If color\_mat has fewer rows than sur\_vox\_mat, you need to decide how to handle it

% One option is to repeat or interpolate colors to match the number of points

% For example, you can use repmat to repeat colors

if size(color\_mat, 1) < n

num\_missing = n - size(color\_mat, 1);

% Repeat the existing colors to match the number of points

color\_mat = repmat(color\_mat, ceil(n/size(color\_mat, 1)), 1);

% Trim the excess rows

color\_mat = color\_mat(1:n, :);

end

sur\_vox\_mat\_ptc = pointCloud(sur\_vox\_mat);

sur\_vox\_mat\_ptc.Color = uint8(color\_mat);

pcwrite(sur\_vox\_mat\_ptc,'dancer\_surf\_2','PLYFormat','ascii');

surf\_mod = pcread('dancer\_surf\_2.ply');

pcshow(surf\_mod);

pcwrite(surf\_mod,"dancer\_surf\_2.ply")

false\_color\_RGB = computeFalseColor(vox\_mat);

writePLYFalseColor(vox\_mat, false\_color\_RGB);

% Step 6: Write a PLY file with false-color RGB values

function writePLYFalseColor(vox\_mat, false\_color\_RGB)

% Create a point cloud object

ptCloud = pointCloud(vox\_mat, 'Color', false\_color\_RGB);

% Write the point cloud to a PLY file

pcwrite(ptCloud, 'output\_false\_color.ply', 'PLYFormat', 'ascii');

end

% Step 5: Determine false-color RGB values for each output 3D point

function false\_color\_RGB = computeFalseColor(vox\_mat)

% Determine range bounding box

X\_min = min(vox\_mat(:, 1));

X\_max = max(vox\_mat(:, 1));

Y\_min = min(vox\_mat(:, 2));

Y\_max = max(vox\_mat(:, 2));

Z\_min = min(vox\_mat(:, 3));

Z\_max = max(vox\_mat(:, 3));

% Calculate normalized coordinates

X\_norm = (vox\_mat(:, 1) - X\_min) / (X\_max - X\_min);

Y\_norm = (vox\_mat(:, 2) - Y\_min) / (Y\_max - Y\_min);

Z\_norm = (vox\_mat(:, 3) - Z\_min) / (Z\_max - Z\_min);

% Assign false-color RGB values

false\_color\_RGB = uint8([255 \* X\_norm, 255 \* Y\_norm, 255 \* Z\_norm]);

end

Results:1. Model using surface voxel

2.False color image using 3D points

3: True color image using 3D points

A person with arms out

Description automatically generated with medium confidence

A graph of a person with orange dots

Description automatically generated

A person in a dress

Description automatically generated

Total number of voxels in the visual hull: 12940

Relative portion of all voxels in the visual hull: 0.0012914

Total number of surface voxels: 1355

Relative portion of all voxels that are surface voxels: 0.00013522

Number of 3D points included in the model: 1573

Number of surface voxels: 1355

The total number of 3D points included in the model (1573) exceeds the number of surface voxels (1355). This discrepancy arises from the fact that each surface voxel contributes multiple points to the final 3D model.

When generating the 3D points from surface voxels, the code calculates points on the surface of each voxel adjacent to an empty voxel in all six cardinal directions (left, right, front, back, top, bottom). This means that a single surface voxel can contribute up to six points to the final model, depending on how many of its faces are adjacent to empty voxels.

As a result, the number of 3D points in the model is typically greater than the number of surface voxels, reflecting the finer level of detail captured by considering multiple points on the surfaces of individual voxels. This finer resolution helps to faithfully represent the shape and contours of the reconstructed object.