Shape From Silhouette Project

By: Nicholas Luis (nml5604@psu.edu). *I have completed this with integrity*
For: Dr. Eapen, Aerospace 497: Signal Processing & Computer Vision

Introduction

The purpose of this project is to learn how to reconstruct a 3D model of an object using pictures of it from different angles. Specifically, this project employs a shape-from-silhouette (SFS) method, which only requires the silhouettes of an object, which is a greek/roman statue in this case. There are 18 images taken of the statue as it is rotated on a rotating platform to capture it from various known angles and positions.

Procedure and Results

The SFS process can be broken down into 5 main steps, which are described below:

Step 1: Silhouette Intensity Thresholding

This first step involved separating the statue from the background. The experimental setup allowed this to be done easily by having the white statue contrast against a black background and table. The intensity threshold was then adjusted until the statue appeared with little noise around it. This was a trial-and-error process that can be seen in Figures 1 through 3 where a **threshold value of 100** was deemed best. Although some aspects of the background are still visible at this value, such as the edge of the table, it was better to ensure that none of the statue was missing.

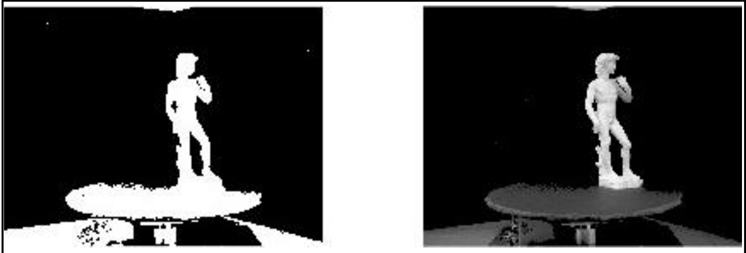


Figure 1. Silhouette of image 18 with a threshold value of 50

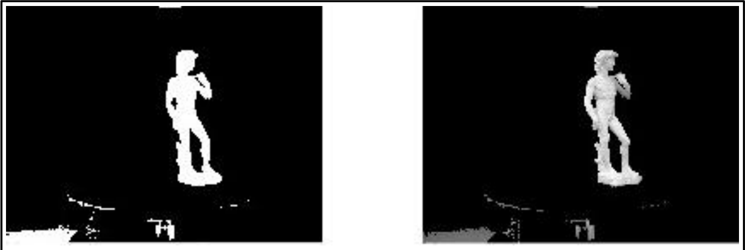


Figure 2. Silhouette of image 18 with a threshold value of 100. Note that statue is seemingly unchanged, but the background is mostly gone

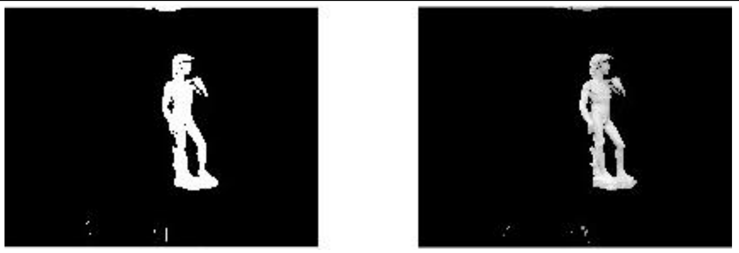


Figure 3. Silhouette of image 18 with a threshold value of 150. Note that the background is now almost entirely gone, but it is missing parts of the statue

Step 2: Coordinate Transformations and Projections

Image generation is only possible if one knows the camera intrinsic and extrinsic parameters for each image to determine where the statue is. This step was already given in the form of a transformation matrix.

Step 3: Defining the Bounding Box and Voxels

Next was to determine the minimum and maximum values of the *bbox* variable to capture the entire statue. Figure 4 shows a very skewed rectangular prism. This was done to determine what the x, y, and z directions are in the image before doing trial and error to properly adjust the bounds of the box.

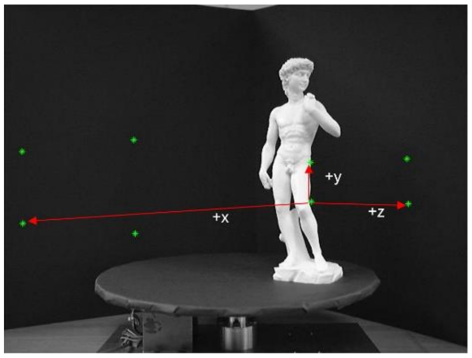
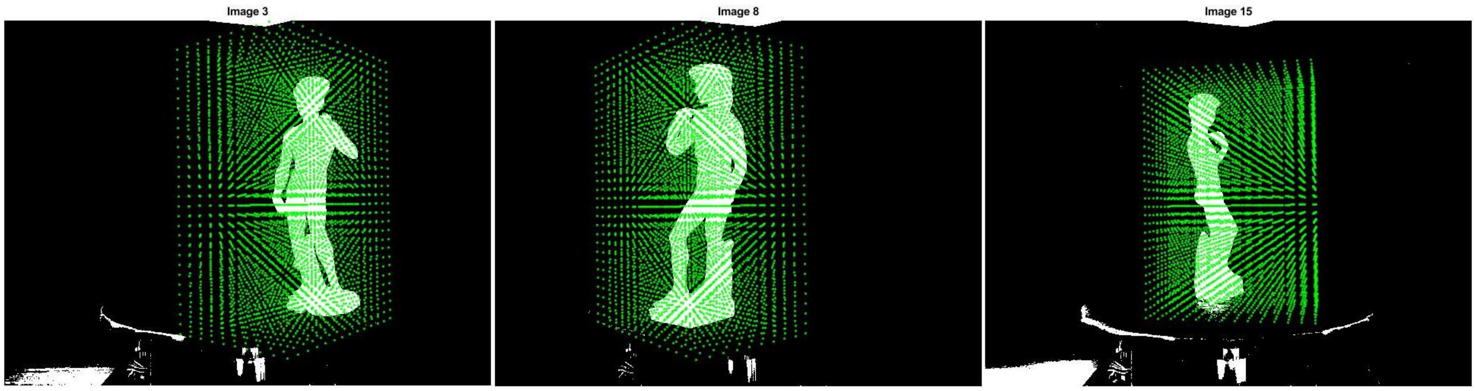


Figure 4. Labeled x, y, and z directions on the image

Figures 5 through 7 shows the centers of each voxel within the bounding box, as viewed from a few different angles. This is just for demonstration purposes so that you can still see the statue, but the **size of the bounding box is unchanged and is defined in the snippet of code below**. The code in its entirety can be found in the appendix.

```
bbox = [-0.25 -2.25 -1.75; 2.25 2.75 1.75]; %[minX minY minZ; maxX maxY maxZ]
```



Figures 5-7. Voxels plotted against the intensity-thresholded images as dots located at their respective center points.

Note that this is just for demonstration purposes so that you can see the statue still. In the actual program, I used a very high number of voxels (3.36×10^7 to be exact) to generate an accurate 3D map. When plotting the centers of the voxels, it appears to be a solid green block, as is seen in Figure 8. This used **200, 600, and 280 voxels in the x, y, and z directions, respectively**. Note that a larger number of voxels were used in the y direction because the way that the statue is rotating means that there is a gap between the layers of voxels in the x-z planes. However, beyond a certain number, there will be no change in the quality of the 3D model because it will be limited by the number of pixels that the object occupies. This limit is difficult to predict because it depends on the resolution of the image, size of the bounding box, and the size of the box after transforming it into pixel coordinates.

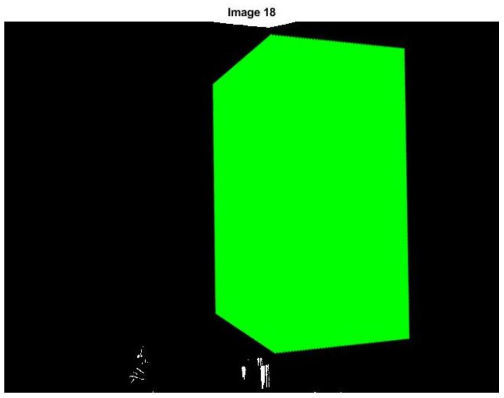


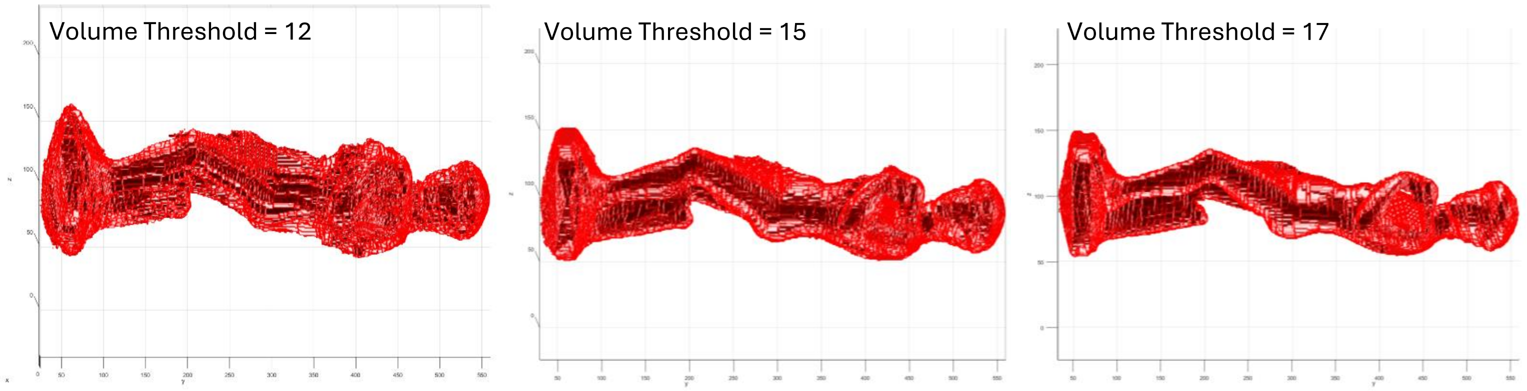
Figure 8. Visualization of the bounding box when using a very large number of voxels

Step 4: Visual Hull Algorithm

There are various ways to determine whether the image occupies a voxel. The method used in this project was to calculate the center point of each voxel, round it to the nearest pixel, and see if that pixel is white or black. If it is white, an occupancy score is added to the corresponding voxel, and this process is repeated for each of the 18 images.

Step 5: Image Generation

Generating the image was simple; Plot the voxels using a meshgrid, but trial and error was once again needed to determine which voxels were rendered. Figure 9 through 11 shows the 3D renderings of the statue at varying volume thresholding values.



Figures 9-11. 3D renders of the statue shown in the y-z plane at various volume thresholding values. Note that quality increases as the threshold increases.

From these, a **volume threshold of 17 produced the most accurate 3D model**. In other words, a particular voxel was only plotted if it was occupied in 17 of the 18 images. Note that a volume threshold of 18 would be ideal, but it produced an empty plot for unknown reasons. It would be better because there is such a high density of voxels, and it is therefore unlikely to mistakenly capture the background or regions not consistently visible across all views. Figure 12 shows the rendered model at various angles at a threshold of 17. Note that the scale is not consistent between the Figures. Though the units are arbitrary, reference the axis labels for its size in each Figure.

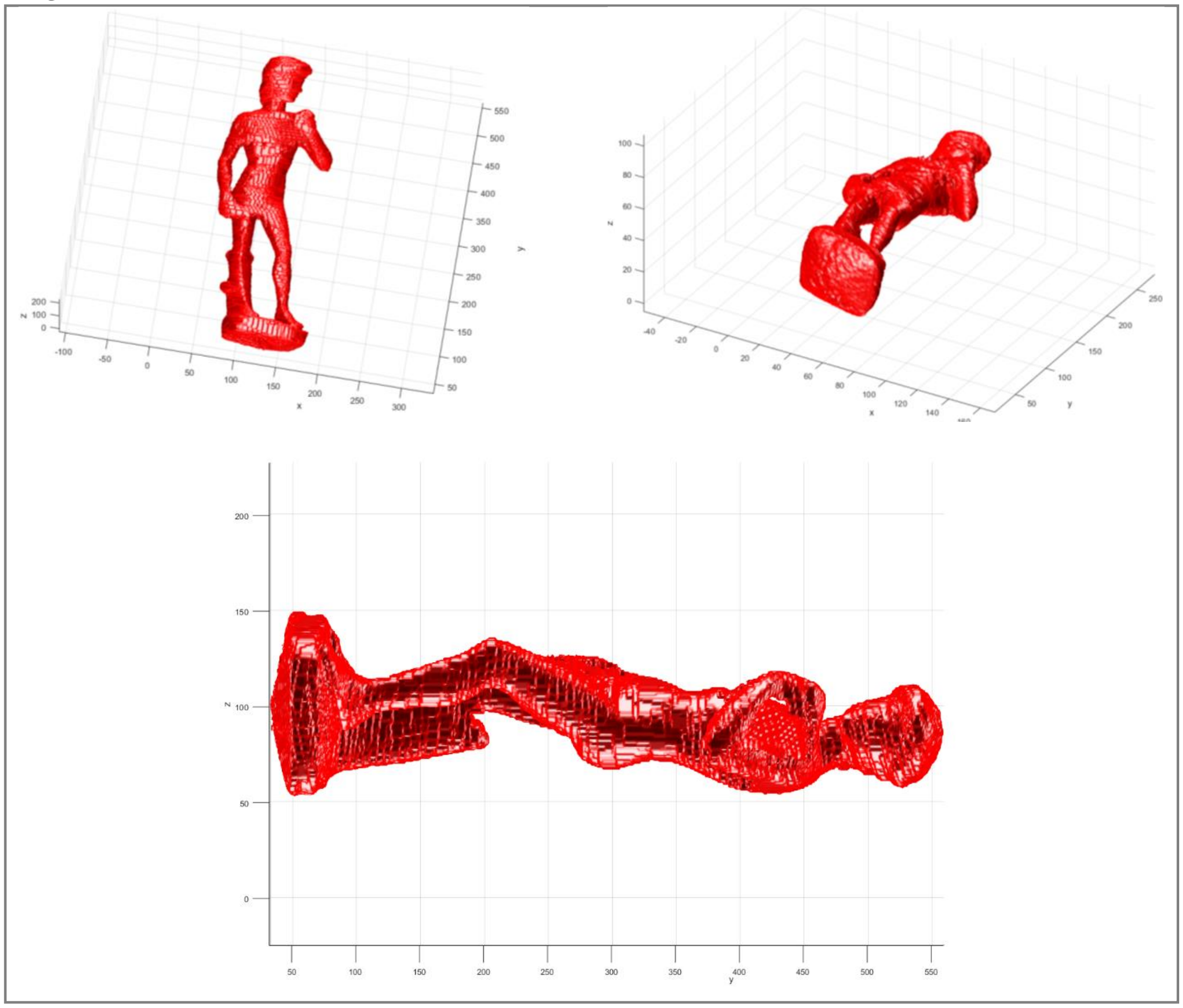


Figure 12. Final 3D Reconstruction of the statue, as seen from various angles

Future Work and Improvements

Although this project worked well, it could still be improved further. For example, the intensity thresholding only worked well with the given data set because the experiment was intentionally designed to have a white object contrast against a black background. A more robust solution would be to employ target tracking of the object in between images to ensure that only the target object is being modeled. Another improvement that could be made would be to image the object from more angles such as a top-down view. This would increase the resolution of the model, which already is limited because it only relies on the silhouette of the object. Similarly, the resolution of the images matters too because the size of the pixels also limits the resolution of the rendered model. Finally, as previously stated, a volume threshold of 18 would be ideal. Running the command *max(max(max(volume)))* shows that there is at least one voxel that is present in all 18 images, but more debugging is needed to determine why it is not plotting correctly.

Conclusion

In short, this project presented a classic Shape-from-Silhouette algorithm to reconstruct a 3D model of a statue using only images of it. It was done using a relatively low silhouette thresholding value to keep lots of data, a high density of voxels to sample that data completely, and a high value for the volume threshold to capture the statue's shape. This combination of parameters allowed for a very accurate 3D model. However, this project was designed for this specific data set. More work is needed to create a more robust system capable of replicating this success across a variety of scenarios.

Appendix

Source Code:

```matlab
%% Metadata
% SPCV Spring 25 - Final Project
% Name: Nicholas Luis
% PSU ID: NML5604
% I have completed this with integrity

% Goals:
%   - Get silhouette of object
%       > adjust threshold value until no part of the statue is missing
%         (alternatively can do object tracking and remove everything else)
%   - Create boundary box filled with voxels
%       > determine the min/max values of the coordinates for the box
%         (done via trial and error)
%       > define the number of voxels in each direction
%   - Determine if a voxel contains our target object
%       > get the center point of each voxel
%       > Transform the voxel center points into 2D image coordinates
%       > Check if the pixel at the coordinate is white in the thresholded
%         image. If so, add an occupancy score to that voxel
%       > repeat for each of the 18 images

clear; clc; close all;

%% Setup

% Use these variables to enable/disable different parts of the script.

loadImages           = true;
displayVolumeCorners = true;
computeVisualHull    = true;
displayIsoSurface    = true;


%% Task B silhouette threshold

% This should be a suitable value between 0 and 255
silhouetteThreshold = 100;% Enter Value Here

%% Task C define bounding box

% It should be as small as possible, but still contain the whole region of
% interest.
bbox = [-0.25 -2.25 -1.75; 2.25 2.75 1.75]; %[minX minY minZ; maxX maxY maxZ]
% Creating evenly spaced voxels
volumeX = 200; % Select # voxels in x
volumeY = 600; % Select # voxels in y
volumeZ = round( volumeX*((bbox(2,3)-bbox(1,3)) / (bbox(2,1)-bbox(1,1))) ); % Create
evenly distributed voxels in the xz plane
volumeThreshold = 16;
```

```
% The volume threshold is used to identify the voxels that have no
% intersection at all. For n (18) images, the volume threshold has to be a
% number less or equal to than n. Discuss in your report the significance
% of this number.

%% Load in images

numCameras = 18;

if loadImages
    % Load silhouette images and projection matrices
    for n=1:numCameras
        % Projection matrices : 3x4 matrix of [R,t].
        % This does not normalize the depth value
        Ps{n} = textread(sprintf('dataSFS/david_%02d.pa',n-1));
        Ps{n} = [eye(3,2) [1 1 1]']*Ps{n};  % add 1 for one-based indices
        % Images
        ims{n} = imread(sprintf('dataSFS/david_%02d.jpg',n-1));
        % Silhouettes: pixels set to one when larger brightness than
        % threshhold
        sils{n} = rgb2gray(ims{n})>silhouetteThreshold;

        figure(1);
        subplot(1,2,1);
        imshow(sils{n});
        title(sprintf('Image %d',n));
        subplot(1,2,2);
        imshow(double(rgb2gray(ims{n}))/255.*sils{n});
        title(sprintf('Image %d',n));
        drawnow;
        pause(0);
    end
end

%% Define transformation from volume to world coordinates.

T = [eye(4,3) [bbox(1,:) 1]'] * ...
    diag([(bbox(2,1)-bbox(1,1))/volumeX ...
          (bbox(2,2)-bbox(1,2))/volumeY ...
          (bbox(2,3)-bbox(1,3))/volumeZ ...
          1]);
T = [1  0 0 0; ...
     0  0 1 0; ...  % flip y and z axes for better display in matlab figure
(isosurface)
     0 -1 0 0; ...
     0  0 0 1] * T;
T = T*[eye(4,3) [-[1 1 1] 1]'];  % subtract 1 for one-based indices


if displayVolumeCorners
    % Draw projection of volume corners.
    for n=1:numCameras
        figure(2);
        hold off;
        imshow(ims{n});
```

```matlab
        hold on;
        corners = [[      0       0       0 1]' ...
                   [      0       0 volumeZ 1]' ...
                   [      0 volumeY       0 1]' ...
                   [      0 volumeY volumeZ 1]' ...
                   [volumeX       0       0 1]' ...
                   [volumeX       0 volumeZ 1]' ...
                   [volumeX volumeY       0 1]' ...
                   [volumeX volumeY volumeZ 1]'];
        pcorners = Ps{n}*T*corners; % Convert to image coordinates
        pcorners = pcorners./repmat(pcorners(3,:),3,1); % Scaling so that third
element is 1
        plot(pcorners(1,:),pcorners(2,:),'g*');
        title(sprintf('Image %d',n));
        drawnow;
        pause(0);

    end
end
%% Task D Visual Hull
if computeVisualHull
    % Define volume. This is used to store the number of observations for each voxel.
    volume = zeros(volumeX,volumeY,volumeZ);

    % calculates center point of each voxel in world coords
    xCorners = linspace( bbox(1,1), bbox(2,1), volumeX+1 );
    yCorners = linspace( bbox(1,2), bbox(2,2), volumeY+1 );
    zCorners = linspace( bbox(1,3), bbox(2,3), volumeZ+1 );

    xCenters = ( xCorners(1:end-1) + xCorners(2:end) )/2;
    yCenters = ( yCorners(1:end-1) + yCorners(2:end) )/2;
    zCenters = ( zCorners(1:end-1) + zCorners(2:end) )/2;

    % creates a N*4 matrix to store the 3 coordinates & a column of 1's
    centers = NaN(volumeX*volumeY*volumeZ, 4); % preallocate for speed
    pixel2voxel = NaN(volumeX*volumeY*volumeZ, 3); % Maps the coordinates of each
centerpoint to a corresponding voxel
    indx = 1;
    for i = 1:volumeX
        for j = 1:volumeY
            for k = 1:volumeZ
                centers(indx, :) = [xCenters(i), yCenters(j), zCenters(k), 1];
                pixel2voxel(indx, :) = [i, j, k];
                indx = indx + 1;
            end
        end
    end

    % Plots the centers of the voxels on the images
    for n = 1:numCameras

        % Converts the center points of each voxel from volume coords to pixel coords
        pcenters = Ps{n}*centers'; % Convert to image coordinates
        pcenters = pcenters./repmat(pcenters(3,:),3,1); % Scaling so that third
element is 1
```

```
        pcenters = round(pcenters); % Round to the nearest pixel

        % Ensures that the transformed coordinates are within the bounds of the image
        [height, width] = size(sils{n});
        for i = 1 : length(pcenters)
            % Checking the u values (columns)
            if ((pcenters(1,i) <= 0) || (pcenters(1,i) > width))
                pcenters(1,i) = NaN;
            end

            % Checking the v values (rows)
            if ((pcenters(2,i) <= 0) || (pcenters(2,i) > height))
                pcenters(2,i) = NaN;
            end
        end

        figure(3);
        imshow(sils{n});
        hold on;
        plot(pcenters(1,:), pcenters(2,:), 'g.');
        title(sprintf('Image %d',n));
        hold off;
        drawnow;
        pause(0);

        % Checks if the voxel center lands on a pixel that is a white or black
        voxelCounter = zeros(length(pcenters), 1);
        for i = 1 : length(pcenters)
            image = sils{n};
            u = pcenters(1,i);
            v = pcenters(2,i);

            if isfinite(u) && isfinite(v)

                % if the voxel center is placed on a white pixel, add it to the
counter
                if (image(v,u) == 1)
                    xIndex = pixel2voxel(i,1);
                    yIndex = pixel2voxel(i,2);
                    zIndex = pixel2voxel(i,3);

                    volume(xIndex, yIndex, zIndex) = 1 + volume(xIndex, yIndex,
zIndex);
                end

            end

        end % loops through each of the voxels

    end % loops through each of the 18 images

end % entire visual hull code

%% Display the isosurface
```

```
% The volume threshold is used to identify the voxels that have no
% intersection at all. For n images, the volume threshold has to be a
% number less or equal to than n. Discuss in your report the significance
% of this number.


if displayIsoSurface
    % display result
    figure(4);
    clf;
    grid on;
    xlabel('x');
    ylabel('y');
    zlabel('z');
    hold on;
    [xMesh, yMesh, zMesh] = meshgrid(1:volumeY,1:volumeX,1:volumeZ);
    pt = patch(isosurface(yMesh, xMesh, zMesh, volume, volumeThreshold));
    set(pt,'FaceColor','red','EdgeColor','none');
    axis equal;
    daspect([volumeX/(bbox(2,1)-bbox(1,1)) volumeY/(bbox(2,2)-bbox(1,2))
volumeZ/(bbox(2,3)-bbox(1,3))]);
    camlight(0,0);
    camlight(180,0);
    camlight(0,90);
    camlight(0,-90);
    lighting phong;
    view(30,30);
end
```