<u>ELL715 Digital Image Processing</u>

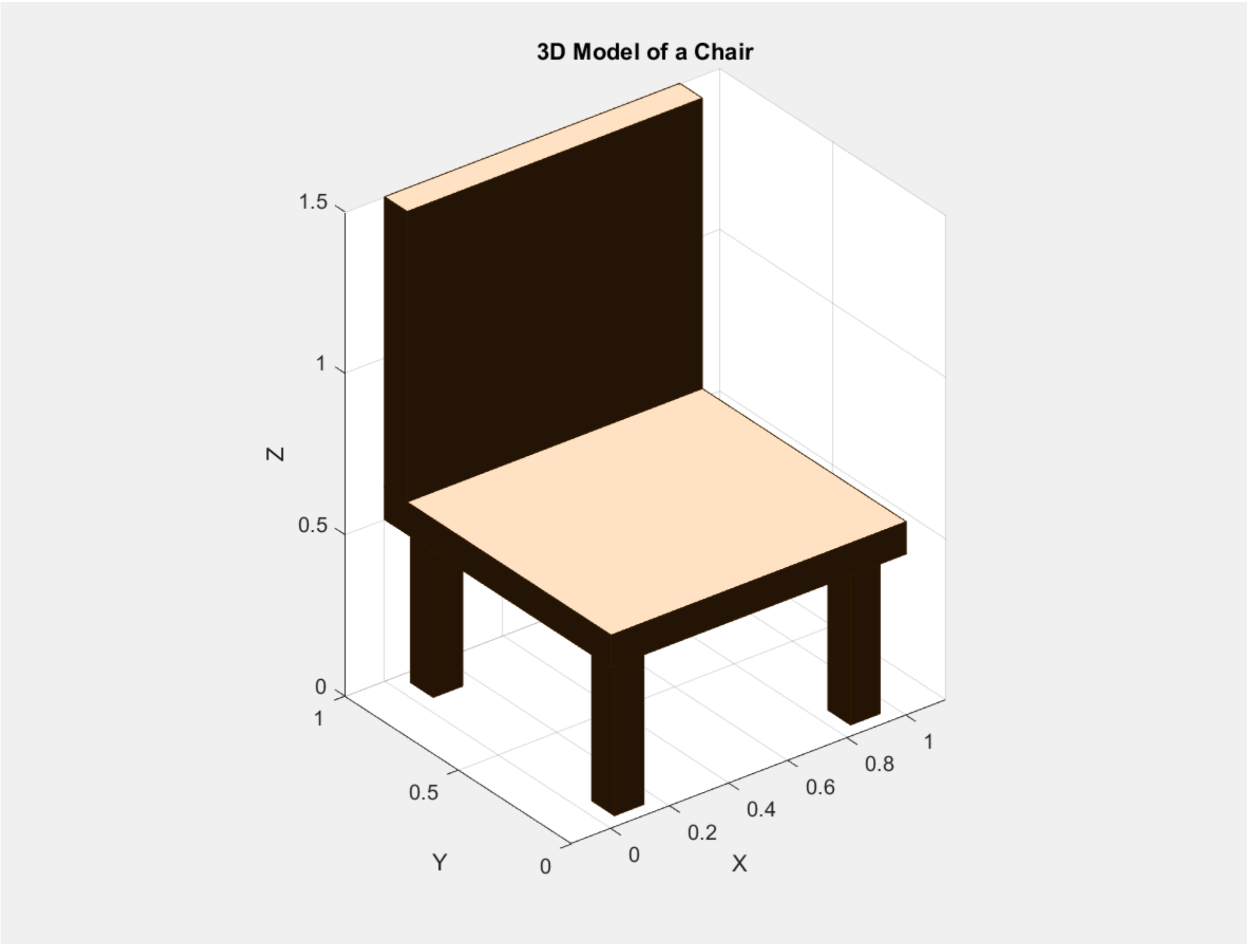
Assignment-1: Virtual Camera Simulation and 3D Reconstruction

~ARYAN VERMA - 2022EE11671

1. 3D Object Definition and Projection

1.1 3D Object

I created a detailed 3D model of a chair using MATLAB. The chair is defined by a set of **vertices** (points in 3D space) and **faces** (surfaces that connect the vertices). The model is composed of several interconnected cuboids to represent the seat, backrest, and four legs. The vertices are represented in **homogeneous coordinates**, which allows for transformations like translation and rotation to be expressed as a single matrix multiplication. The homogeneous coordinates of a 3D point are $(x,\ y,\ z,\ 1)$.



1.2 Virtual Cameras and Projection

Two virtual cameras with different positions `[0, 0, 5]` & `[-7.07, 0, 0]` and orientations were created. The projection of a 3D point from a camera onto a 2D image plane is determined by the camera's **projection matrix**. This matrix is a product of two components:

- Intrinsic Matrix (K): This matrix describes the camera's internal properties, such as focal length `[500, 500]` and principal point `[320, 240]`. For this simulation, the intrinsic matrix was defined as:

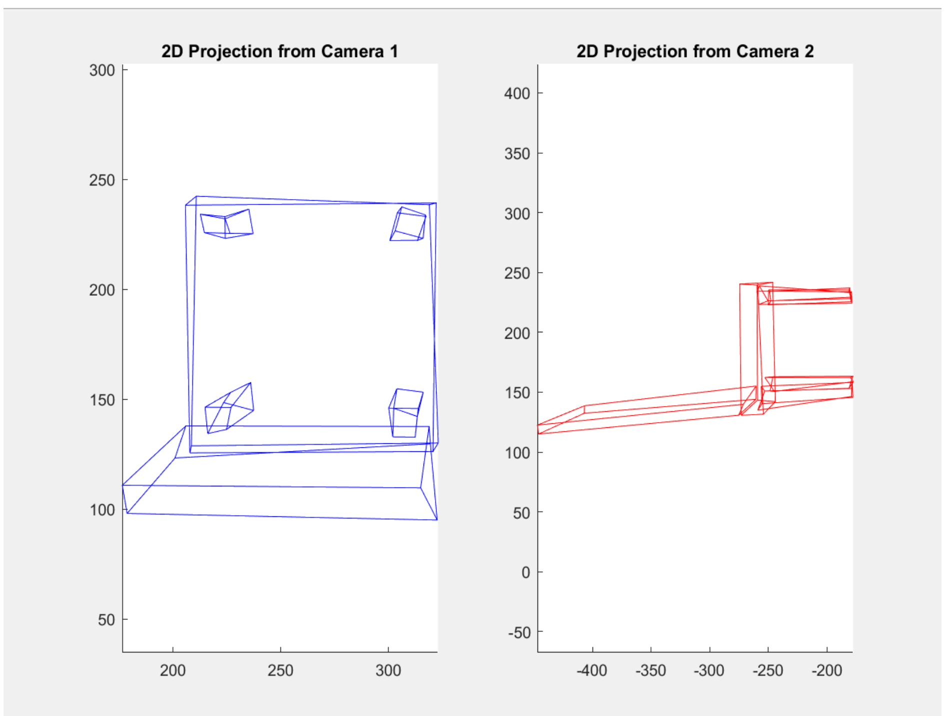
$$\begin{bmatrix} 500 & 0 & 320 \\ 0 & 500 & 240 \\ 0 & 0 & 1 \end{bmatrix}$$

- **Extrinsic Matrix (**$[R|t]$**)**: This matrix describes the camera's position and orientation in the world. It consists of a rotation matrix (R) and a translation vector (t).

The projection of a **3D point X_{world} to a 2D point X_{image}** is given by the equation:

$$X_{image} = K[R|t]X_{world} = P*X_{world}$$

The 3D vertices of the chair were projected onto two separate 2D views, one for each camera. **Gaussian noise** was added to these 2D projected points to simulate real-world measurement inaccuracies



2. 3D Reconstruction from 2 Cameras

2.1 Direct Linear Transformation (DLT)

The **Direct Linear Transformation (DLT)** algorithm was used to reconstruct the 3D chair model from the two 2D views. The core idea is to find the 3D point that minimizes the reprojection error across all camera views. For each 3D point, its corresponding 2D projections in both camera images are known. These relationships can be expressed as a system of linear equations.

For a 3D point $\boldsymbol{X}$ = `[x, y, z, 1]`^{T} and its 2D projection $\boldsymbol{x}_{proj}$ = `[u, v, 1]`^{T} from a camera with projection matrix P, the relationship is given by:

$\boldsymbol{x}_{proj}\ \boldsymbol{x}\ (\boldsymbol{PX}) = \boldsymbol{0}$ *(Cross product)*

This cross-product can be expanded into a set of linear equations. By combining the equations from both cameras, a system of linear equations is formed:

AX = 0

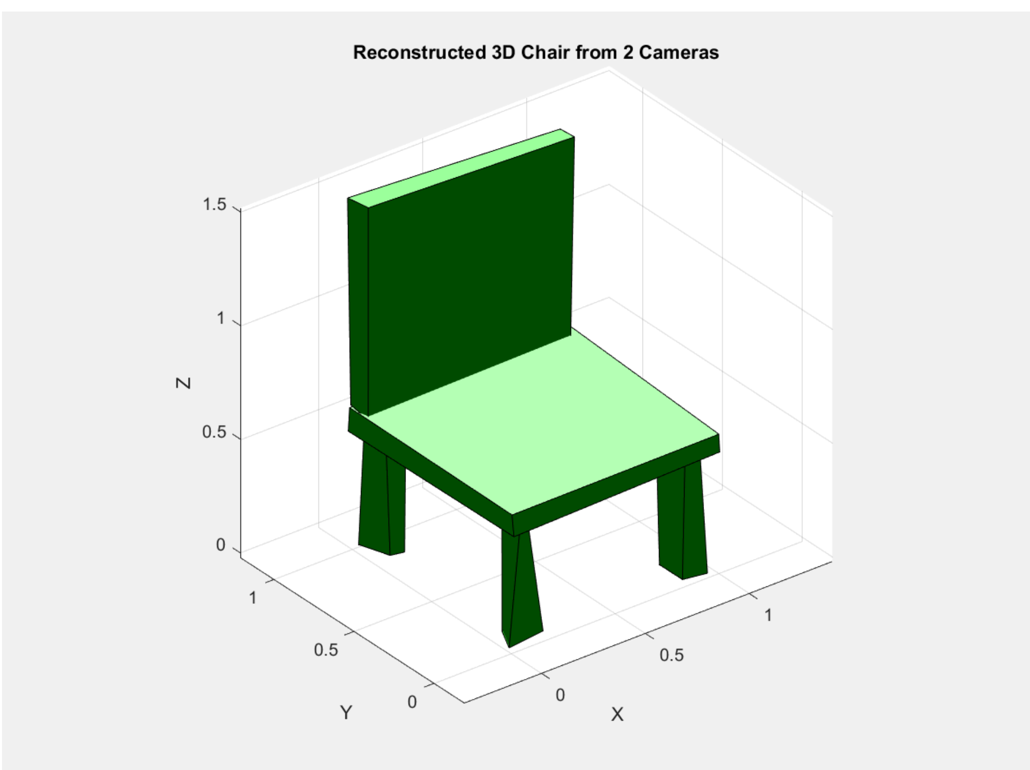
This system is solved for X (the 3D point in homogeneous coordinates) using **Singular Value Decomposition (SVD)**. The solution is the column of the V matrix corresponding to the smallest singular value.

2.2 Reconstruction Error

The reconstruction error was calculated by finding the average Euclidean distance between the original 3D points (v) and the reconstructed 3D points (vreconstructed). The formula used is:

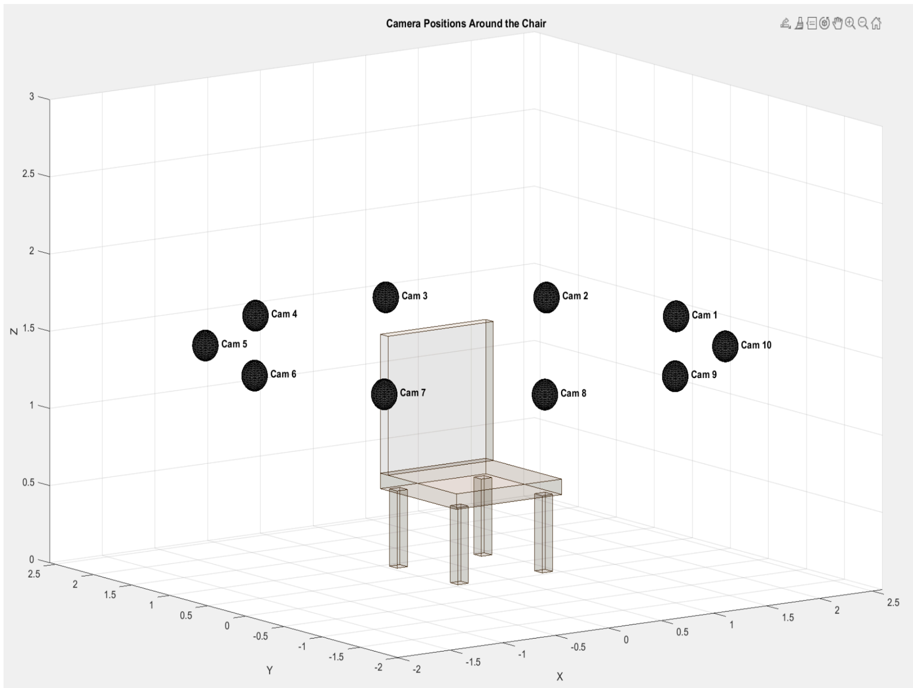
Error= $$\frac{1}{N}\sum_{i=1}^{N}\sqrt{\left(v_{reconstructed,\,i}\ -\ v_i\right)^2}$$

The reconstruction error with two cameras was calculated to be approximately **0.021681**.



3. Impact of Number of Cameras

The reconstruction process was repeated by increasing the number of cameras from 2 to 10. The cameras were positioned along a circular path around the object to ensure a variety of viewpoints. The simulation was run multiple times for each number of cameras, and the average error was calculated to mitigate the effect of the random noise added to the projections.



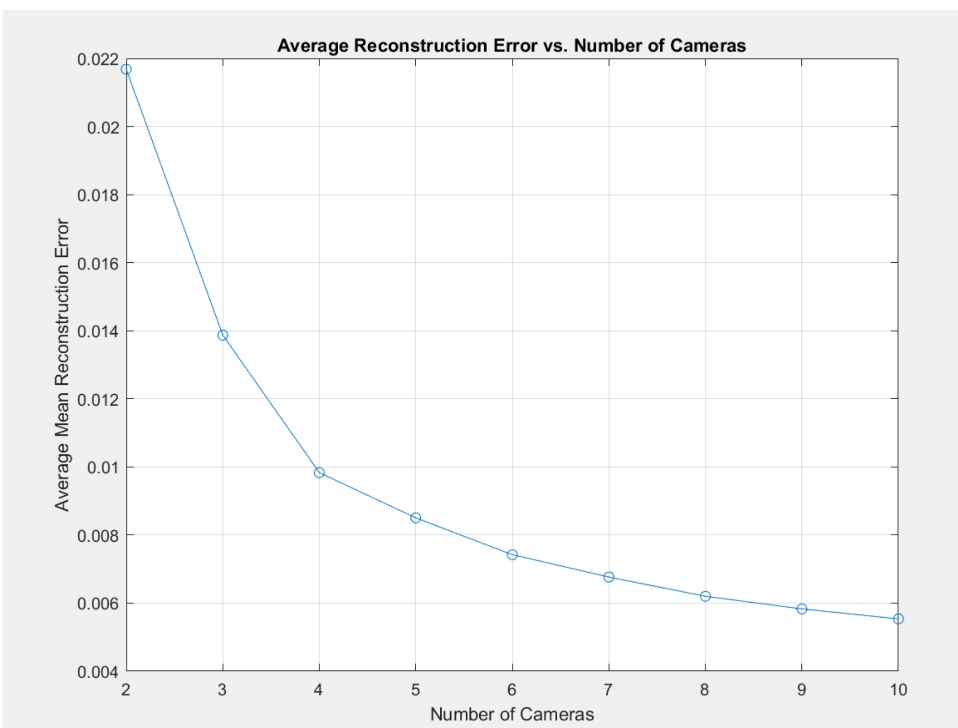
3.1 Results and Analysis

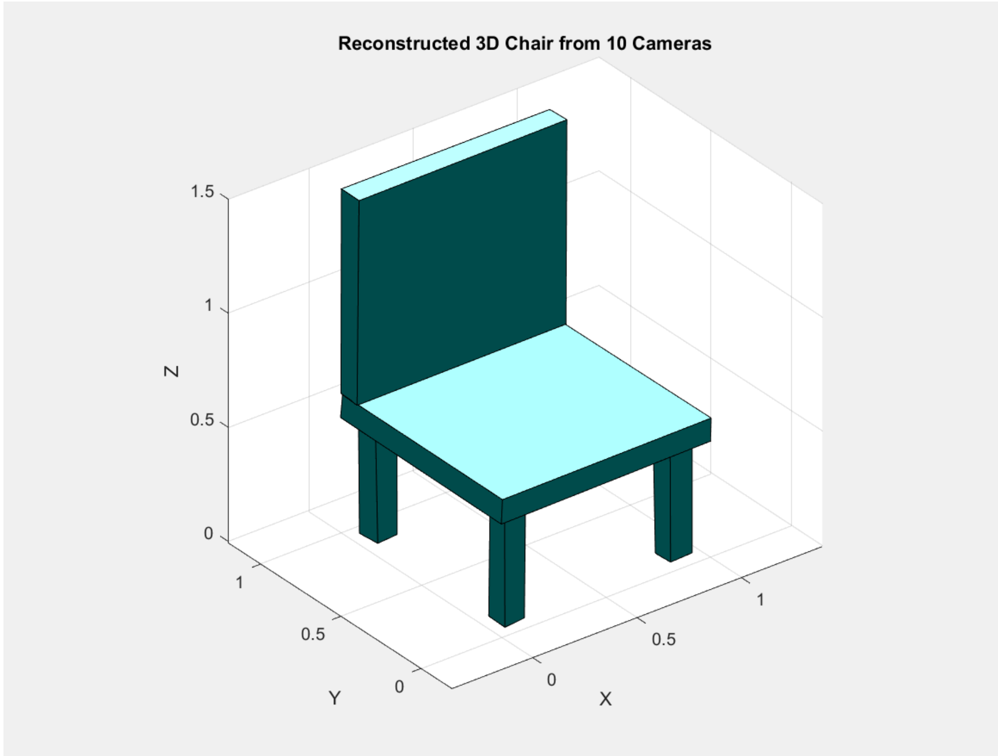
The results show a clear trend: as the number of cameras increases, the average reconstruction error decreases. The improvement is significant when moving from 2 to 3 or 4 cameras, but the rate of improvement slows down as more cameras are added. This is because adding more cameras provides additional geometric constraints, making the system of equations for DLT more robust and less susceptible to noise.

Here is a summary of the average reconstruction errors:

Number of Cameras	Average Mean Reconstruction Error
2	0.021681

3	0.013872
4	0.009832
5	0.008503
6	0.007422
7	0.006766
8	0.006200
9	0.005830
10	0.005537





4. Co-located Cameras

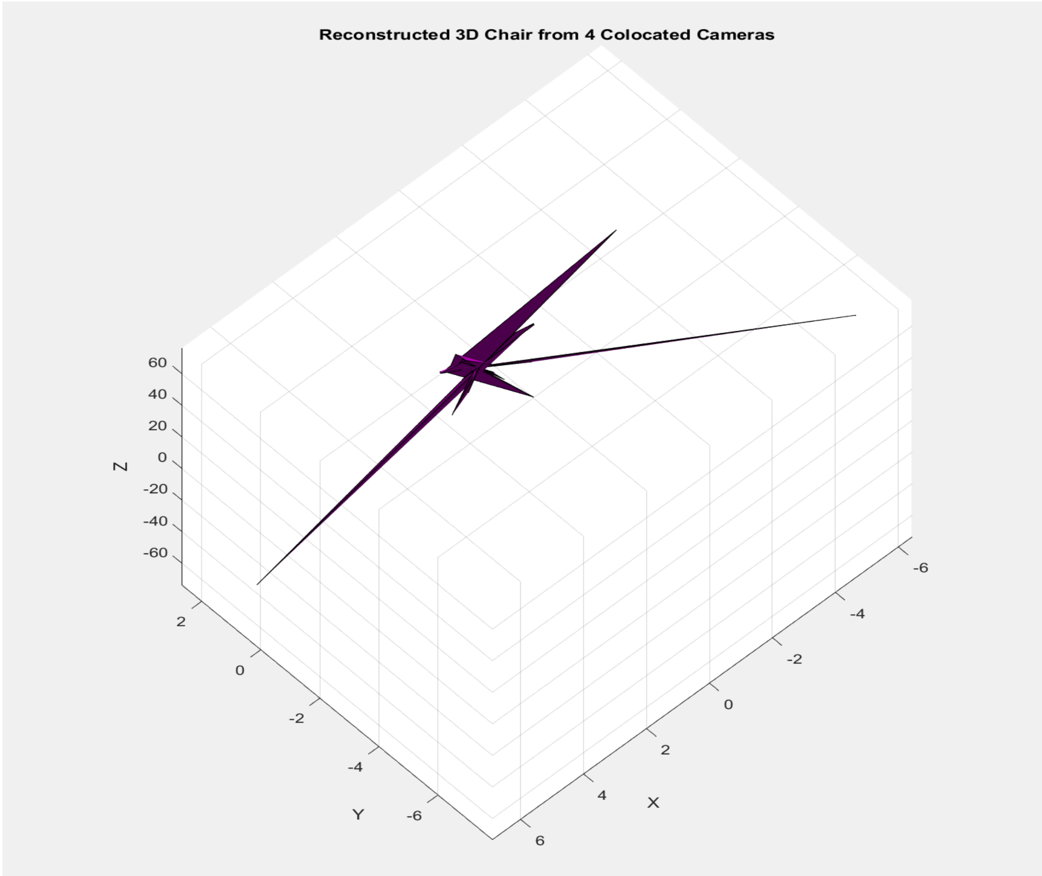
4.1 Effect of Co-located Cameras

The assignment also asked to analyze what happens when cameras are nearly co-located. In this simulation, four cameras were placed at almost the same position but with very small rotations relative to each other.

The reconstruction error with four co-located cameras was calculated to be **7.097218**. This value is significantly higher than the error achieved with a distributed set of cameras.

4.2 Results

When cameras are co-located, their viewpoints are nearly identical. This means that the 2D projections from these cameras are very similar, even with the small rotations introduced. This geometric configuration leads to an ill-conditioned system of equations when attempting to triangulate the 3D points. The linear equations from each camera become highly dependent, providing redundant information rather than new geometric constraints. As a result, the DLT algorithm becomes less stable and more sensitive to noise, leading to a much higher reconstruction error compared to cameras with distinct viewpoint



5. Codebase

```
clc;        % Clear the command window
clear all;  % Clear all variables from the workspace
close all;  % Close all open figures
%% Part 1: Generate a More Detailed 3D Model of a Chair with Better Colors
% Define vertices and faces for a more detailed chair
% Modeling it with several interconnected cuboids
% [x, y, z] coordinates for a better chair model
v = [
   % Seat
   0.0 0.0 0.5; 1.0 0.0 0.5; 1.0 1.0 0.5; 0.0 1.0 0.5; % Bottom of seat
   0.0 0.0 0.6; 1.0 0.0 0.6; 1.0 1.0 0.6; 0.0 1.0 0.6; % Top of seat
   % Backrest
   0.0 0.9 0.6; 1.0 0.9 0.6; 1.0 0.9 1.5; 0.0 0.9 1.5; % Bottom of backrest
   0.0 1.0 0.6; 1.0 1.0 0.6; 1.0 1.0 1.5; 0.0 1.0 1.5; % Top of backrest
   % Legs (modeled as thin cuboids)
   % Leg 1 (Front-Left)
   0.05 0.05 0; 0.15 0.05 0; 0.15 0.15 0; 0.05 0.15 0; % Bottom
   0.05 0.05 0.5; 0.15 0.05 0.5; 0.15 0.15 0.5; 0.05 0.15 0.5; % Top
   % Leg 2 (Front-Right)
   0.85 0.05 0; 0.95 0.05 0; 0.95 0.15 0; 0.85 0.15 0; % Bottom
   0.85 0.05 0.5; 0.95 0.05 0.5; 0.95 0.15 0.5; 0.85 0.15 0.5; % Top
   % Leg 3 (Back-Right)
   0.85 0.85 0; 0.95 0.85 0; 0.95 0.95 0; 0.85 0.95 0; % Bottom
   0.85 0.85 0.5; 0.95 0.85 0.5; 0.95 0.95 0.5; 0.85 0.95 0.5; % Top
```

```
   % Leg 4 (Back-Left)
   0.05 0.85 0; 0.15 0.85 0; 0.15 0.95 0; 0.05 0.95 0; % Bottom
   0.05 0.85 0.5; 0.15 0.85 0.5; 0.15 0.95 0.5; 0.05 0.95 0.5; % Top
];
% Define faces for the new chair model
f = [
   % Seat faces
   1 2 3 4; 5 6 7 8; 1 5 6 2; 2 6 7 3; 3 7 8 4; 4 8 5 1;
   % Backrest faces
   9 10 11 12; 9 10 14 13; 10 11 15 14; 11 12 16 15; 12 9 13 16;
   % Leg 1 faces
   17 18 19 20; 21 22 23 24; 17 21 22 18; 18 22 23 19; 19 23 24 20; 20 24 21 17;
   % Leg 2 faces
   25 26 27 28; 29 30 31 32; 25 29 30 26; 26 30 31 27; 27 31 32 28; 28 32 29 25;
   % Leg 3 faces
   33 34 35 36; 37 38 39 40; 33 37 38 34; 34 38 39 35; 35 39 40 36; 36 40 37 33;
   % Leg 4 faces
   41 42 43 44; 45 46 47 48; 41 45 46 42; 42 46 47 43; 43 47 48 44; 44 48 45 41;
];
% Create the 3D plot of the chair
figure('Position', [100, 100, 800, 600]);
patch('Vertices', v, 'Faces', f, 'FaceColor', [0.5, 0.3, 0.1], 'EdgeColor', [0.2, 0.1, 0]);
light('Position',[1 1 1]);
title('3D Model of a Chair');
xlabel('X'); ylabel('Y'); zlabel('Z');
axis equal; grid on; view(3);
%% Part 2: Projections from 2 Cameras and Reconstruction
% Camera parameters
K = [500 0 320; 0 500 240; 0 0 1]; % Intrinsic matrix
v_homogeneous = [v, ones(size(v, 1), 1)]';
% Define two camera poses (Extrinsic parameters)
R1 = eye(3); t1 = [0; 0; -5];
R2 = [cos(pi/4) 0 sin(pi/4); 0 1 0; -sin(pi/4) 0 cos(pi/4)]; t2 = [5; 0; -5];
P1 = K * [R1, t1];
P2 = K * [R2, t2];
% Project 3D points to 2D
projected_v1 = P1 * v_homogeneous;
projected_v2 = P2 * v_homogeneous;
projected_v1 = projected_v1 ./ projected_v1(3, :);
projected_v2 = projected_v2 ./ projected_v2(3, :);
% Add Gaussian noise
noise_std = 2;
noisy_v1 = projected_v1(1:2, :) + noise_std * randn(2, size(v, 1));
noisy_v2 = projected_v2(1:2, :) + noise_std * randn(2, size(v, 1));
% Plot the 2D projections using lines
figure('Position', [100, 100, 800, 600]);
subplot(1, 2, 1);
hold on;
for i = 1:size(f, 1)
   current_face_indices = f(i, :);
   projected_points_x = [noisy_v1(1, current_face_indices), noisy_v1(1,
current_face_indices(1))];
```

```
   projected_points_y = [noisy_v1(2, current_face_indices), noisy_v1(2,
current_face_indices(1))];
   line(projected_points_x, projected_points_y, 'Color', 'b');
end
title('2D Projection from Camera 1'); axis equal; hold off;
subplot(1, 2, 2);
hold on;
for i = 1:size(f, 1)
   current_face_indices = f(i, :);
   projected_points_x = [noisy_v2(1, current_face_indices), noisy_v2(1,
current_face_indices(1))];
   projected_points_y = [noisy_v2(2, current_face_indices), noisy_v2(2,
current_face_indices(1))];
   line(projected_points_x, projected_points_y, 'Color', 'r');
end
title('2D Projection from Camera 2'); axis equal; hold off;
% Reconstruct the 3D chair using DLT (Direct Linear Transformation)
v_reconstructed = zeros(size(v));
for i = 1:size(v, 1)
   A = [
       noisy_v1(1, i) * P1(3, :) - P1(1, :);
       noisy_v1(2, i) * P1(3, :) - P1(2, :);
       noisy_v2(1, i) * P2(3, :) - P2(1, :);
       noisy_v2(2, i) * P2(3, :) - P2(2, :)
   ];
   [~, ~, V] = svd(A);
   X = V(:, end);
   v_reconstructed(i, :) = X(1:3)' / X(4);
end
% Plot the reconstructed 3D model with planes
figure('Position', [100, 100, 800, 600]);
patch('Vertices', v_reconstructed, 'Faces', f, 'FaceColor', 'green', 'EdgeColor', 'black');
light('Position',[1 1 1]);
title('Reconstructed 3D Chair from 2 Cameras');
xlabel('X'); ylabel('Y'); zlabel('Z');
axis equal; grid on; view(3);
% Calculate and print the reconstruction error
error_2cam = mean(sqrt(sum((v_reconstructed - v).^2, 2)));
fprintf('Reconstruction error with 2 cameras: %f\n', error_2cam);
%% Part 3: Reconstruction with 2 to 10 Cameras (Fixed Camera Placement with Visualization)
num_cameras = 2:10;
reconstruction_errors_avg = zeros(size(num_cameras));
num_simulations = 10; % Run the simulation multiple times to average out noise
% Define a circular path for the cameras to ensure good geometry
radius = 2.0;
center = mean(v);
theta = linspace(0, 2*pi, 11); % 11 positions to cover the range 2-10
figure('Position', [100, 100, 800, 600]);
hold on;
patch('Vertices', v, 'Faces', f, 'FaceColor', [0.5, 0.3, 0.1], 'EdgeColor', [0.2, 0.1, 0],
'FaceAlpha', 0.1);
light('Position',[1 1 1]);
```

```
title('Camera Positions Around the Chair');
xlabel('X'); ylabel('Y'); zlabel('Z');
axis([-2 2 -2 2 0 3]); % Adjusted axis limits for better view
grid on; view(3);
% Define an "eyeball" camera shape
[sphere_x, sphere_y, sphere_z] = sphere(20);
eyeball_r = 0.1;
sphere_v = [sphere_x(:)*eyeball_r, sphere_y(:)*eyeball_r, sphere_z(:)*eyeball_r];
sphere_f = convhull(sphere_v); % Use convhull for a valid face list
[iris_x, iris_y, iris_z] = cylinder(eyeball_r*0.5, 20);
iris_v = [iris_x(:), iris_y(:), iris_z(:)];
iris_f = [1:20; 21:40]';
iris_v(:,3) = iris_v(:,3)*0.1;
cam_positions = zeros(3, length(num_cameras));
for k = 1:length(num_cameras)
   n_cam = num_cameras(k);
   total_error = 0;
   for sim = 1:num_simulations
       P_mats = cell(1, n_cam);
       noisy_projections = cell(1, n_cam);

       for i = 1:n_cam
           cam_x = center(1) + radius * cos(theta(i));
           cam_y = center(2) + radius * sin(theta(i));
           cam_z = center(3) + 1;
           cam_pos = [cam_x; cam_y; cam_z];

           if sim == 1
               cam_positions(:, i) = cam_pos;
           end

           direction_vector = cam_pos - center';
           direction_vector = direction_vector / norm(direction_vector);
           up_vector = [0; 0; 1];

           right_vector = cross(up_vector, direction_vector);
           right_vector = right_vector / norm(right_vector);

           new_up_vector = cross(direction_vector, right_vector);
           new_up_vector = new_up_vector / norm(new_up_vector);

           R_structured = [right_vector, new_up_vector, direction_vector]';

           P_mats{i} = K * [R_structured, -R_structured * cam_pos];

           projected_v = P_mats{i} * v_homogeneous;
           projected_v = projected_v ./ projected_v(3, :);
           noisy_projections{i} = projected_v(1:2, :) + noise_std * randn(2, size(v, 1));
       end
       v_reconstructed_multi = zeros(size(v));
       for j = 1:size(v, 1)
           A = [];
```

```
            for i = 1:n_cam
                P_i = P_mats{i};
                proj_i = noisy_projections{i};
                A = [
                    A;
                    proj_i(1, j) * P_i(3, :) - P_i(1, :);
                    proj_i(2, j) * P_i(3, :) - P_i(2, :);
                ];
            end
            [~, ~, V] = svd(A);
            X = V(:, end);
            v_reconstructed_multi(j, :) = X(1:3)' / X(4);
        end
        total_error = total_error + mean(sqrt(sum((v_reconstructed_multi - v).^2, 2)));
    end
    reconstruction_errors_avg(k) = total_error / num_simulations;
end
% Plot camera positions as eyeball shapes with labels
for i = 0:size(cam_positions, 2)-1
    cam_pos_i = cam_positions(:, i+1);

    % Sclera (white part)
    patch('Vertices', sphere_v + cam_pos_i', 'Faces', sphere_f, 'FaceColor', 'w', 'EdgeColor',
'k', 'FaceAlpha', 0.8);

    % Iris (colored part)
    patch('Vertices', iris_v + cam_pos_i', 'Faces', iris_f, 'FaceColor', [0.2, 0.4, 0.6],
'EdgeColor', 'none', 'FaceAlpha', 0.9);

    % Pupil (black dot)
    plot3(cam_pos_i(1), cam_pos_i(2), cam_pos_i(3), 'k.', 'MarkerSize', 15);

    % Label the camera
    text(cam_pos_i(1) + 0.15, cam_pos_i(2), cam_pos_i(3), sprintf('Cam %d', i+1), 'FontWeight',
'bold');
end
hold off;
figure('Position', [100, 100, 800, 600]);
plot(num_cameras, reconstruction_errors_avg, '-o');
title('Average Reconstruction Error vs. Number of Cameras');
xlabel('Number of Cameras'); ylabel('Average Mean Reconstruction Error');
grid on;
fprintf('\nAverage Reconstruction errors for 2 to 10 cameras (over 10 simulations):\n');
fprintf('  Number of Cameras | Error\n');
fprintf('  ------------------|-------\n');
for i = 1:length(num_cameras)
    fprintf('  %17d | %f\n', num_cameras(i), reconstruction_errors_avg(i));
end
%% Reconstructed image for 10-camera case
figure('Position', [100, 100, 800, 600]);
patch('Vertices', v_reconstructed_multi, 'Faces', f, 'FaceColor', 'cyan', 'EdgeColor', 'black');
light('Position',[1 1 1]);
```

```matlab
title('Reconstructed 3D Chair from 10 Cameras');
xlabel('X'); ylabel('Y'); zlabel('Z');
axis equal; grid on; view(3);
%% Part 4: Reconstruction with 4 Colocated Cameras (Improved Visibility)
R1_coloc = eye(3); t1_coloc = [0; 0; -5];
P1_coloc = K * [R1_coloc, t1_coloc];
P_coloc = cell(1, 4);
noisy_projections_coloc = cell(1, 4);
% Four colocated cameras with small rotations
for i = 1:4
   t_coloc = t1_coloc;
   % Small rotations to provide triangulation
   R_coloc = rotx((i-1)*0.2) * roty((i-1)*0.2) * rotz((i-1)*0.2);
   P_coloc{i} = K * [R_coloc, t_coloc];
   projected_v_coloc = P_coloc{i} * v_homogeneous;
   projected_v_coloc = projected_v_coloc ./ projected_v_coloc(3, :);
   noisy_projections_coloc{i} = projected_v_coloc(1:2, :) + noise_std * randn(2, size(v, 1));
end
v_reconstructed_coloc = zeros(size(v));
for i = 1:size(v, 1)
   A = [];
   for j = 1:4
       P_j = P_coloc{j};
       proj_j = noisy_projections_coloc{j};
       A = [
           A;
           proj_j(1, i) * P_j(3, :) - P_j(1, :);
           proj_j(2, i) * P_j(3, :) - P_j(2, :);
       ];
   end
   [~, ~, V] = svd(A);
   X = V(:, end);
   v_reconstructed_coloc(i, :) = X(1:3)' / X(4);
end
figure('Position', [100, 100, 800, 600]);
patch('Vertices', v_reconstructed_coloc, 'Faces', f, 'FaceColor', 'magenta', 'EdgeColor', 'k');
light('Position',[1 1 1]);
title('Reconstructed 3D Chair from 4 Colocated Cameras');
xlabel('X'); ylabel('Y'); zlabel('Z');
axis equal; grid on; view(45, 30); % Adjusted view angle for better visibility
error_coloc = mean(sqrt(sum((v_reconstructed_coloc - v).^2, 2)));
fprintf('\nReconstruction error with 4 colocated cameras: %f\n', error_coloc);
%% Helper Functions for Rotation Matrices
% These functions are created to replace rotx, roty, and rotz
% which are part of specific toolboxes that may not be installed.
function R_x = rotx(theta_deg)
   theta_rad = deg2rad(theta_deg);
   R_x = [1 0 0; 0 cos(theta_rad) -sin(theta_rad); 0 sin(theta_rad) cos(theta_rad)];
end
function R_y = roty(theta_deg)
   theta_rad = deg2rad(theta_deg);
   R_y = [cos(theta_rad) 0 sin(theta_rad); 0 1 0; -sin(theta_rad) 0 cos(theta_rad)];
```

```
end
function R_z = rotz(theta_deg)
   theta_rad = deg2rad(theta_deg);
   R_z = [cos(theta_rad) -sin(theta_rad) 0; sin(theta_rad) cos(theta_rad) 0; 0 0 1];
end
```