%% Run TreeQSM with LAS File using Lidar Toolbox

% Simple script to load a LAS file and run TreeQSM

% Requires MATLAB Lidar Toolbox

clear all; close all; clc;

%% Configuration

% CHANGE THIS to your LAS file path

las\_file = 'C:\Users\c72liu\OneDrive - University of Waterloo\hullet\individualtree\_hullet\high\_resolution\_single\_tree\highres\_208\_off\_individual\MVCA\_208\_GS0007.las';

% TreeQSM parameters

downsample\_threshold = 200000;  % Downsample if more than this many points

filter\_point\_cloud = true;      % Apply filtering to remove noise

auto\_parameters = true;         % Use automatic parameter selection

%% Step 1: Setup

addpath(genpath('src'));

disp('TreeQSM paths added');

%% Step 2: Load LAS file

disp(['Loading LAS file: ', las\_file]);

try

    % Create LAS file reader

    lasReader = lasFileReader(las\_file);

    % Read point cloud

    ptCloud = readPointCloud(lasReader);

    P = double(ptCloud.Location);

    % Debug: Check initial dimensions

    fprintf('Initial point cloud size: %d x %d\n', size(P));

    if size(P, 2) ~= 3

        error('Point cloud should have 3 columns (X,Y,Z), but has %d columns', size(P, 2));

    end

    % Remove invalid points

    valid = all(isfinite(P), 2);

    P = P(valid, :);

    % Debug: Check dimensions after filtering invalid points

    fprintf('After removing invalid points: %d x %d\n', size(P));

    disp(['Loaded ', num2str(size(P,1)), ' valid points']);

    % Display point cloud info

    fprintf('\nPoint Cloud Statistics:\n');

    fprintf('X range: %.2f to %.2f m\n', min(P(:,1)), max(P(:,1)));

    fprintf('Y range: %.2f to %.2f m\n', min(P(:,2)), max(P(:,2)));

    fprintf('Z range: %.2f to %.2f m\n', min(P(:,3)), max(P(:,3)));

catch ME

    error(['Failed to load LAS file: ', ME.message]);

end

%% Step 3: Preprocessing

% Center the point cloud

P\_mean = mean(P);

P = P - P\_mean;

disp('Point cloud centered');

% Debug: Check dimensions after centering

fprintf('After centering: %d x %d\n', size(P));

% Downsample if needed

if size(P,1) > downsample\_threshold

    fprintf('Downsampling from %d to %d points...\n', size(P,1), downsample\_threshold);

    idx = randperm(size(P,1), downsample\_threshold);

    P = P(idx, :);

    % Debug: Check dimensions after downsampling

    fprintf('After downsampling: %d x %d\n', size(P));

end

%% Step 4: Filter point cloud (optional)

if filter\_point\_cloud

    disp('Filtering point cloud...');

    % Define filter parameters

    filter\_inputs.filter.k = 15;            % k-nearest neighbors

    filter\_inputs.filter.nsigma = 2;        % Standard deviation threshold

    filter\_inputs.filter.radius = 0;        % Radius-based (0 = disabled)

    filter\_inputs.filter.ncomp = 5;         % Minimum component size

    filter\_inputs.filter.PatchDiam1 = 0.05;

    filter\_inputs.filter.BallRad1 = 0.075;

    filter\_inputs.filter.EdgeLength = 0;    % Voxel downsampling (0 = disabled)

    filter\_inputs.filter.plot = 0;

    P\_original = P;

    % Debug: Check what filtering returns

    filter\_result = filtering(P, filter\_inputs);

    fprintf('Filtering returned: %d x %d\n', size(filter\_result));

    % Check if filtering returned the expected format

    if size(filter\_result, 2) == 3

        P = filter\_result;

        fprintf('Filtering removed %d points (%.1f%%)\n', ...

            size(P\_original,1) - size(P,1), ...

            (size(P\_original,1) - size(P,1))/size(P\_original,1)\*100);

    else

        % If filtering didn't return expected format, skip filtering

        warning('Filtering function returned unexpected format. Skipping filtering step.');

        fprintf('Expected 3 columns, got %d columns\n', size(filter\_result, 2));

        P = P\_original;

    end

    % Debug: Check dimensions after filtering

    fprintf('After filtering: %d x %d\n', size(P));

end

%% Step 5: Visualize point cloud

% Validate point cloud dimensions before plotting

if size(P, 2) ~= 3

    error('Point cloud must have 3 columns (X,Y,Z) for plotting, but has %d columns', size(P, 2));

end

if size(P, 1) == 0

    error('Point cloud is empty after processing');

end

figure('Name', 'LAS Point Cloud', 'Position', [100, 100, 800, 600]);

plot3(P(:,1), P(:,2), P(:,3), '.', 'MarkerSize', 1, 'Color', [0.2 0.6 0.2]);

title(sprintf('%s\n%d points', las\_file, size(P,1)));

xlabel('X (m)'); ylabel('Y (m)'); zlabel('Z (m)');

axis equal; view(3); grid on;

rotate3d on;

%% Step 6: Setup TreeQSM parameters

if auto\_parameters

    % Automatic parameter selection based on tree size

    disp('Using automatic parameter selection...');

    inputs = define\_input(P, 2, 2, 2);  % 2 values for each parameter

    % Validate automatic parameters

    if isnan(inputs.PatchDiam1(1)) || isnan(inputs.PatchDiam2Min(1)) || isnan(inputs.PatchDiam2Max(1))

        warning('Automatic parameter selection failed (returned NaN). Using manual parameters instead.');

        auto\_parameters = false;  % Switch to manual mode

    else

        fprintf('Automatic parameters:\n');

        fprintf('  PatchDiam1: %.3f, %.3f\n', inputs.PatchDiam1);

        fprintf('  PatchDiam2Min: %.3f, %.3f\n', inputs.PatchDiam2Min);

        fprintf('  PatchDiam2Max: %.3f, %.3f\n', inputs.PatchDiam2Max);

    end

end

if ~auto\_parameters

    % Manual parameters - suitable for medium-large trees

    disp('Using manual parameters...');

    % Calculate reasonable parameters based on point cloud size

    point\_density = size(P,1) / (range(P(:,1)) \* range(P(:,2)) \* range(P(:,3)));

    if point\_density > 50000  % High density

        inputs.PatchDiam1 = [0.06 0.10];

        inputs.PatchDiam2Min = [0.02 0.025];

        inputs.PatchDiam2Max = [0.05 0.08];

    else  % Medium density

        inputs.PatchDiam1 = [0.08 0.12];

        inputs.PatchDiam2Min = [0.025 0.035];

        inputs.PatchDiam2Max = [0.07 0.10];

    end

    inputs.BallRad1 = inputs.PatchDiam1 + 0.015;

    inputs.BallRad2 = inputs.PatchDiam2Max + 0.01;

    fprintf('Manual parameters:\n');

    fprintf('  PatchDiam1: %.3f, %.3f\n', inputs.PatchDiam1);

    fprintf('  PatchDiam2Min: %.3f, %.3f\n', inputs.PatchDiam2Min);

    fprintf('  PatchDiam2Max: %.3f, %.3f\n', inputs.PatchDiam2Max);

end

% Set other parameters

inputs.OnlyTree = 1;

inputs.Tria = 0;

inputs.Dist = 1;

inputs.MinCylRad = 0.0025;

inputs.ParentCor = 1;

inputs.TaperCor = 1;

inputs.GrowthVolCor = 0;

% Output settings

[~, filename, ~] = fileparts(las\_file);

inputs.name = filename;

inputs.tree = 1;

inputs.model = 1;

inputs.savemat = 1;

inputs.savetxt = 0;

inputs.plot = 1;

inputs.disp = 1;

% Create results folder

if ~exist('results', 'dir')

    mkdir('results');

end

%% Step 7: Run TreeQSM

fprintf('\n========== Running TreeQSM ==========\n');

tic;

% Run QSM reconstruction

QSM = treeqsm(P, inputs);

elapsed = toc;

fprintf('QSM reconstruction completed in %.1f seconds\n', elapsed);

%% Step 8: Save and Display results

if ~isempty(QSM)

    % Manual save to results directory

    save\_filename = sprintf('QSM\_%s\_t1\_m1.mat', filename);

    save\_path = fullfile('results', save\_filename);

    save(save\_path, 'QSM');

    fprintf('\n========== Tree Measurements ==========\n');

    fprintf('Tree Height:        %.2f m\n', QSM.treedata.TreeHeight);

    fprintf('Trunk Volume:       %.1f L\n', QSM.treedata.TrunkVolume \* 1000);

    fprintf('Branch Volume:      %.1f L\n', QSM.treedata.BranchVolume \* 1000);

    fprintf('Total Volume:       %.1f L\n', QSM.treedata.TotalVolume \* 1000);

    fprintf('DBH:                %.1f cm\n', QSM.treedata.DBHqsm \* 100);

    fprintf('Number of Branches: %d\n', QSM.treedata.NumberBranches);

    fprintf('Max Branch Order:   %d\n', QSM.treedata.MaxBranchOrder);

    if isfield(QSM, 'pmdistance')

        fprintf('\n========== Model Quality ==========\n');

        fprintf('Mean point-model distance: %.1f mm\n', QSM.pmdistance.mean \* 1000);

        fprintf('Std deviation:             %.1f mm\n', QSM.pmdistance.std \* 1000);

    end

    fprintf('\n========== Output Files ==========\n');

    fprintf('Results saved in: results/QSM\_%s\_t1\_m1.mat\n', filename);

    % Plot final QSM

    figure('Name', 'Final QSM Model', 'Position', [950, 100, 800, 600]);

    plot\_cylinder\_model(QSM.cylinder, 20, 0.8, 2);

    title('TreeQSM Model - Colored by Branch Order');

    axis equal; view(3);

else

    warning('QSM reconstruction failed. Check your point cloud data.');

end

disp(' ');

disp('Script completed successfully!');