%% Fractal Dimension Analysis System

function fractal\_dimension\_analysis()

% Initialize environment

clc; clearvars; close all;

warning('off', 'Images:initSize:adjustingMag');

rng(20230312);

try

%% Module 1: Dual Image Loading

[img\_orig, colorbar\_img] = image\_loader();

if isempty(img\_orig) || isempty(colorbar\_img)

error('Image loading failed');

end

%% Module 2: Interactive Threshold Configuration

[thresh, swap\_color] = interactive\_threshold\_setup(img\_orig, colorbar\_img);

[max\_power, frame\_delay, batch\_mode] = advanced\_param\_setup();

%% Module 3: Core Analysis Process

binary\_img = process\_binarization(img\_orig, colorbar\_img, thresh, swap\_color);

fig\_handle = create\_analysis\_interface(img\_orig, binary\_img, colorbar\_img, thresh, swap\_color);

[box\_sizes, counts, D] = dynamic\_box\_counting(binary\_img, max\_power, frame\_delay, batch\_mode, fig\_handle);

visualize\_fractal(log(1./box\_sizes), log(counts), D, fig\_handle, counts, box\_sizes);

catch ME

errordlg(['System exception: ' ME.message], 'Execution interrupted');

end

end

%% Image Loader

function [img\_orig, colorbar\_img] = image\_loader()

% Main image loading

[fname, path] = uigetfile({'\*.jpg;\*.png;\*.bmp;\*.tif', 'Image files'},...

'Select image for analysis', 'MultiSelect', 'off');

if isequal(fname, 0)

img\_orig = []; colorbar\_img = [];

return;

end

img\_orig = im2double(imread(fullfile(path, fname)));

% Colorbar loading

[cname, cpath] = uigetfile({'\*.jpg;\*.png;\*.bmp;\*.tif', 'Image files'},...

'Select corresponding colorbar', 'MultiSelect', 'off');

colorbar\_img = im2double(imread(fullfile(cpath, cname)));

end

%% Interactive Threshold Setup Interface

function [thresh, swap\_state] = interactive\_threshold\_setup(img\_orig, colorbar\_img)

fig = figure('Position', [500 400 1100 600], 'Name', 'Threshold Configurator', ...

'NumberTitle', 'off', 'MenuBar', 'none', 'Resize', 'off', ...

'CloseRequestFcn', @close\_callback);

% Image display area

ax\_main = axes('Parent', fig, 'Position', [0.05 0.3 0.6 0.6], 'Tag', 'main\_ax');

img\_preview = imshow(imfuse(img\_orig, zeros(size(img\_orig)), 'montage'), 'Parent', ax\_main);

title(ax\_main, 'Real-time Binarization Preview', 'FontSize', 10);

% Colorbar display area

ax\_colorbar = axes('Parent', fig, 'Position', [0.7 0.3 0.25 0.6], 'Tag', 'colorbar\_ax');

imshow(colorbar\_img, 'Parent', ax\_colorbar);

% Add scale to original color map

caxis(ax\_colorbar, [0 1]); % Set min and max values for colorbar

colormap(ax\_colorbar, 'jet');

colorbar(ax\_colorbar, 'eastoutside');

% Create red threshold line

[h, w, ~] = size(colorbar\_img);

if h > w

thresh\_line = line(ax\_colorbar, [1 w], [1 1], 'Color', 'r', 'LineWidth', 2);

orientation = 'vertical';

else

thresh\_line = line(ax\_colorbar, [1 1], [1 h], 'Color', 'r', 'LineWidth', 2);

orientation = 'horizontal';

end

% Interactive controls

slider = uicontrol('Style', 'slider', 'Position', [650 50 200 25], ...

'Min', 0, 'Max', 1, 'Value', 0.5, 'Tag', 'slider', 'Callback', @update\_preview);

thresh\_value\_display = uicontrol('Style', 'text', 'Position', [860 50 40 25], ...

'String', '0.50', 'Tag', 'thresh\_value', 'FontSize', 10);

uicontrol('Style', 'checkbox', 'Position', [650 20 100 25], ...

'String', 'Invert Colors', 'Tag', 'swap', 'Callback', @update\_preview);

uicontrol('Style', 'pushbutton', 'Position', [650 100 100 30], ...

'String', 'Confirm Parameters', 'Callback', @confirm\_params, ...

'BackgroundColor', [0.85 0.95 1]);

% Data storage

data = struct('thresh', 0.5, 'swap', false, 'completed', false);

guidata(fig, data);

uiwait(fig);

function update\_preview(src, ~)

data = guidata(fig);

if strcmp(get(src, 'Tag'), 'slider')

data.thresh = get(src, 'Value');

else

data.swap = get(src, 'Value');

end

set(thresh\_value\_display, 'String', sprintf('%.2f', 1-data.thresh));

binary = process\_binarization(img\_orig, colorbar\_img, data.thresh, data.swap);

set(img\_preview, 'CData', imfuse(img\_orig, binary, 'montage'));

% Adjust threshold line position calculation

if strcmp(orientation, 'vertical')

% Vertical orientation: Y coordinate = (thresh) \* h

yPos = (data.thresh) \* h;

set(thresh\_line, 'YData', [yPos yPos]);

else

% Horizontal orientation: X coordinate = thresh \* w

xPos = data.thresh \* w;

set(thresh\_line, 'XData', [xPos xPos]);

end

guidata(fig, data);

end

function confirm\_params(~, ~)

data = guidata(fig);

data.completed = true;

guidata(fig, data);

set(fig, 'Visible', 'off');

uiresume(fig);

end

function close\_callback(~, ~)

if ishghandle(fig)

delete(fig);

end

end

data = guidata(fig);

thresh = data.thresh;

swap\_state = data.swap;

end

%% Dynamic Box Counting Algorithm

function [box\_sizes, counts, D] = dynamic\_box\_counting(binary\_img, max\_power, frame\_delay, batch\_mode, fig)

% Parameter validation

if ~isvalid(fig)

fig = figure('Position', [100 100 1600 700], 'Name', 'Fractal Dimension Analysis System');

end

% Initialize axes

ax = findobj(fig, 'Tag', 'dynamic\_ax');

if isempty(ax) || ~isvalid(ax)

ax = axes('Parent', fig, 'Position', [0.02 0.05 0.45 0.3],...

'Tag', 'dynamic\_ax', 'NextPlot', 'add');

imshow(binary\_img, 'Parent', ax);

title(ax, 'Box Counting Dynamic Demonstration', 'FontSize', 10);

end

% Calculation parameters

[h, w] = size(binary\_img);

max\_size = min([h, w, 2^floor(log2(max(h, w)))]);

box\_sizes = unique(round(2.^linspace(1, log2(max\_size), max\_power)));

counts = zeros(size(box\_sizes));

% Main calculation loop

for k = 1:length(box\_sizes)

s = box\_sizes(k);

count = 0;

% Delete rectangles from previous size

delete(findobj(ax, 'Type', 'rectangle'));

% Traverse image and perform box counting

for y = 1:s:h

for x = 1:s:w

x\_end = min(x + s - 1, w);

y\_end = min(y + s - 1, h);

% Check if any pixel is 0 (background) in this region

if any(~binary\_img(y:y\_end, x:x\_end), 'all')

count = count + 1;

rectangle(ax, 'Position', [x, y, x\_end - x, y\_end - y],...

'EdgeColor', [0.9 0.2 0.2], 'LineWidth', 1.2);

end

% Dynamic image update (pause if not in batch mode)

if ~batch\_mode

drawnow;

pause(frame\_delay);

end

end

end

% Update box count

counts(k) = count;

% Display current box count

text(0.5, 0.95, sprintf('Box Size: %d, Count: %d', s, count),...

'HorizontalAlignment', 'center', 'VerticalAlignment', 'top',...

'Color', 'white', 'FontSize', 12, 'FontWeight', 'bold', 'Parent', ax);

% Skip display delay in batch mode

if batch\_mode

drawnow;

else

pause(frame\_delay);

end

end

% Calculate fractal dimension

D = polyfit(log(1 ./ box\_sizes), log(counts), 1);

D = D(1);

% Display final result

disp('Fractal Dimension:');

disp(D);

% Add fractal dimension display to window

text(0.5, 0.85, sprintf('Fractal Dimension: %.4f', D),...

'HorizontalAlignment', 'center', 'VerticalAlignment', 'top',...

'Color', 'white', 'FontSize', 12, 'FontWeight', 'bold', 'Parent', ax);

end

%% Result Visualization

function visualize\_fractal(log\_s, log\_N, D, fig, counts, box\_sizes)

% Axes management

if ~isvalid(fig)

fig = figure;

end

ax = findobj(fig, 'Tag', 'result\_ax');

if isempty(ax) || ~isvalid(ax)

ax = axes('Parent', fig, 'Position', [0.52 0.05 0.45 0.3],...

'Tag', 'result\_ax', 'NextPlot', 'add');

end

% Data visualization

cla(ax);

plot(ax, log\_s, log\_N, 'o-', 'Color', [0 0.45 0.75], 'LineWidth', 2,...

'MarkerFaceColor', [0.8 0.9 1], 'MarkerSize', 8);

fit\_line = plot(ax, log\_s, polyval(polyfit(log\_s, log\_N, 1), log\_s),...

'--', 'Color', [0.95 0.3 0.1], 'LineWidth', 2.5);

% Graph annotations

xlabel(ax, 'ln(1/s)');

ylabel(ax, 'ln(N)');

title(ax, sprintf('Fractal Dimension D = %.3f', D), 'FontSize', 12);

grid(ax, 'on');

legend(ax, fit\_line, ['Fit Slope: ' num2str(D, '%.3f')], 'Location', 'southeast');

% Data output

fprintf('Box Sizes (px)\n');

fprintf('%2d\n', box\_sizes);

fprintf('Coverage Counts\n');

fprintf('%2d\n', counts);

end

%% Advanced Parameter Setup

function [max\_power, frame\_delay, batch\_mode] = advanced\_param\_setup()

% Dialog box base parameters

dlgWidth = 400;

dlgHeight = 300;

dlg = dialog('Units', 'pixels',...

'Position', [600 400 dlgWidth dlgHeight],...

'Name', 'Advanced Parameter Setup',...

'WindowStyle', 'modal',...

'Color', [0.95 0.95 0.95],...

'CloseRequestFcn', @close\_callback); % Add close callback

% Initialize parameter storage structure

params = struct(...

'max\_power', 8,...

'frame\_delay', 0.1,...

'batch\_mode', true);

% Store control handles

handles = struct();

%% Create interface controls

% Unified control parameters

controlMargin = 20; % Margin

rowHeight = 40; % Row height

labelWidth = 140; % Label width

editWidth = 80; % Input box width

% Calculate vertical positions (layout from top)

yPos = dlgHeight - controlMargin - rowHeight;

%% 1. Box size quantity setting

handles.txtPower = uicontrol(dlg,...

'Style', 'text',...

'Position', [controlMargin yPos+8 labelWidth 20],...

'String', 'Box Sizes Count (4-12):',...

'HorizontalAlignment', 'left',...

'FontSize', 9);

handles.editPower = uicontrol(dlg,...

'Style', 'edit',...

'Position', [controlMargin+labelWidth+10 yPos editWidth 30],...

'String', num2str(params.max\_power),...

'Tooltip', 'Controls number of boxes at different scales',...

'FontSize', 9,...

'BackgroundColor', [1 1 1],...

'Callback', @edit\_callback);

%% 2. Animation frame delay setting

yPos = yPos - rowHeight;

handles.txtDelay = uicontrol(dlg,...

'Style', 'text',...

'Position', [controlMargin yPos+8 labelWidth 20],...

'String', 'Frame Delay (seconds):',...

'HorizontalAlignment', 'left',...

'FontSize', 9);

handles.editDelay = uicontrol(dlg,...

'Style', 'edit',...

'Position', [controlMargin+labelWidth+10 yPos editWidth 30],...

'String', num2str(params.frame\_delay),...

'Tooltip', 'Controls animation refresh speed (0.01-2 sec)',...

'FontSize', 9,...

'BackgroundColor', [1 1 1],...

'Callback', @edit\_callback);

%% 3. Batch mode setting

yPos = yPos - rowHeight;

handles.chkBatch = uicontrol(dlg,...

'Style', 'checkbox',...

'Position', [controlMargin yPos+5 labelWidth+editWidth+10 30],...

'String', 'Enable Batch Processing Mode',...

'Value', params.batch\_mode,...

'Tooltip', 'Disable animation to speed up processing',...

'FontSize', 9);

%% Confirm button

btnWidth = 80;

btnHeight = 30;

handles.btnConfirm = uicontrol(dlg,...

'Style', 'pushbutton',...

'Position', [(dlgWidth-btnWidth)/2 controlMargin btnWidth btnHeight],...

'String', 'Confirm',...

'FontSize', 10,...

'FontWeight', 'bold',...

'BackgroundColor', [0.85 0.94 0.83],...

'Callback', @confirm\_callback);

% Store data

guidata(dlg, struct('params', params, 'handles', handles));

% Wait for window to close

uiwait(dlg);

% Return final parameters

max\_power = params.max\_power;

frame\_delay = params.frame\_delay;

batch\_mode = params.batch\_mode;

%% Callback functions

function edit\_callback(src, ~)

data = guidata(dlg);

try

value = str2double(get(src, 'String'));

switch src

case data.handles.editPower

data.params.max\_power = validate(value, 4, 12, 8);

case data.handles.editDelay

data.params.frame\_delay = validate(value, 0.01, 2, 0.1);

end

catch

errordlg('Invalid input value, check parameter range', 'Parameter Error');

end

guidata(dlg, data);

end

function confirm\_callback(~, ~)

data = guidata(dlg);

% Final validation

try

data.params.max\_power = validate(...

str2double(get(data.handles.editPower, 'String')),...

4, 12, 8);

data.params.frame\_delay = validate(...

str2double(get(data.handles.editDelay, 'String')),...

0.01, 2, 0.1);

data.params.batch\_mode = logical(...

get(data.handles.chkBatch, 'Value'));

catch ME

errordlg(sprintf('Parameter error: %s', ME.message), 'Configuration Error');

return;

end

% Update parameters and close

params = data.params;

delete(dlg);

end

function close\_callback(~, ~)

% Ensure safe closure

if ishandle(dlg)

delete(dlg);

end

uiresume(dlg);

end

%% Validation function

function result = validate(value, min\_val, max\_val, default)

if isnan(value) || value < min\_val || value > max\_val

errordlg(sprintf('Parameter range: %.2f to %.2f', min\_val, max\_val),...

'Validation Error');

result = default;

else

result = value;

end

end

end

%% Binarization Processing Core

function binary\_img = process\_binarization(img\_orig, colorbar\_img, thresh, swap\_color)

% Extract color sequence

[h, w, ~] = size(colorbar\_img);

if h > w

color\_seq = squeeze(colorbar\_img(:, floor(w/2), :));

else

color\_seq = squeeze(colorbar\_img(floor(h/2), :, :));

end

% Build color map

img\_flat = reshape(img\_orig, [], 3);

distances = pdist2(img\_flat, color\_seq, 'euclidean');

[~, idx] = min(distances, [], 2);

% Generate binary matrix

color\_length = max(h, w);

positions = (idx-1)/(color\_length-1);

binary\_img = reshape(positions >= thresh, size(img\_orig,1), []);

% Color inversion

if swap\_color

binary\_img = ~binary\_img;

end

end

%% Create Analysis Interface

function fig = create\_analysis\_interface(img\_orig, binary\_img, colorbar\_img, thresh, swap\_state)

% Reuse existing window

fig = findobj('Type', 'figure', 'Name', 'Fractal Dimension Analysis System');

if isempty(fig)

fig = figure('Position', [100 100 1600 700], 'Color', [0.95 0.95 0.95],...

'Name', 'Fractal Dimension Analysis System', 'NumberTitle', 'off');

else

clf(fig); % Clear existing content

set(fig, 'Position', [100 100 1600 700]); % Reset window position

end

% Main image display

ax\_main = axes('Parent', fig, 'Position', [0.02 0.4 0.45 0.5], 'Tag', 'main\_ax');

imshow(imfuse(img\_orig, binary\_img, 'montage'), 'Parent', ax\_main);

title(ax\_main, sprintf('Original vs Binarized\\nThreshold=%.2f | Inverted=%d', 1-thresh, swap\_state),...

'FontSize', 10, 'Interpreter', 'none');

% Colorbar display

ax\_colorbar = axes('Parent', fig, 'Position', [0.52 0.4 0.45 0.5], 'Tag', 'colorbar\_ax');

[h, w, ~] = size(colorbar\_img);

if h > w

imshow(colorbar\_img, 'Parent', ax\_colorbar);

line(ax\_colorbar, [1 w], [thresh\*h thresh\*h], 'Color', 'r', 'LineWidth', 2);

else

imshow(colorbar\_img, 'Parent', ax\_colorbar);

line(ax\_colorbar, [thresh\*w thresh\*w], [1 h], 'Color', 'r', 'LineWidth', 2);

end

end

%% Helper Function

function val = clamp(val, min\_val, max\_val)

val = max(min\_val, min(val, max\_val));

end