Project Title: Generating Mandala Designs In MATLAB

By: Anirudh Jain, Ayush Pandey, Chitranshi Kumre

Abstract

This project presents a mathematical and visual approach to generating artistic mandala patterns using MATLAB. By employing core concepts of linear algebra-specifically, transformation matrices-we automate symmetrical designs with different base geometries (using parametric equations) and user-defined parameters. The project also introduces a graphical user interface (GUI) that enables users to dynamically alter shapes, apply matrix transformations, and visualize eigenvectors (for shear transformation). This report elaborates on the mathematical foundations, MATLAB implementation strategies, and the interactive elements of the GUI developed.

Keywords: MATLAB, GUI, matrix transformations.

TABLE OF FIGURES

1. INTRODUCTION

Mandala patterns are intricate and symmetrical figures, often used in spiritual and artistic contexts. This project focuses on generating such patterns programmatically using MATLAB, while using concepts of linear algebra. The aim is to visualize mathematical transformations like rotation, reflection, shear, and scaling in an artistic context, reinforcing theoretical knowledge through practical application.

2. OBJECTIVES:

2.1. Main Objective

The main objective of out project is to demonstrate different concepts of linear algebra by *generating beautiful Mandala designs in MATLAB* by using core concepts of algebra like *rotation, reflection, scaling* etc.

2.2. Secondary Objective

Integrating a *simple GUI* for user friendly interactions to generate different designs by changing input values like the number of sectors(petal) or layers etc.

3. METHODOLOGY

We used MATLAB to generate mandala designs through the following steps:

3.1. **Base Shape Generation:**
Shapes like petals, polygons, and rose curves were defined using parametric equations and stored as 2D vectors in matrix form.

3.2. **Transformation Application:**
We applied linear transformations using 2×2 matrices — including rotation (radial symmetry), reflection, shear, and scaling (layer effect). These were combined through matrix multiplication to create composite transformations.

3.3. **Interactive GUI Development:**
A programmatic GUI was built using uicontrol () and figure () functions, allowing users to input values like sector count, shear factors, and shape type.

3.4. **Plotting & Visualization:**
Final shapes were drawn using plot () or fill (), with options to change background colour or display eigenvectors of the shear matrix for deeper insight.

4. CORE CONCEPTS OF LINEAR ALGEBRA

4.1. Vector Representation

Each petal is defined as a set of vectors in 2D space, constructed from parametric equations. These vectors represent discrete (x, y) coordinates along the petal curve and are stored in a matrix form to facilitate linear algebra operations.

4.2. Matrix Multiplication

Matrix multiplication forms the backbone of all transformations in this project. Each petal shape is defined as a series of 2D points (vectors), and transformations are applied using 2×2 matrices. It is a binary operation that produces a matrix from two matrices. For matrix multiplication, the number of columns in the first matrix must be equal to the number of rows in the second matrix.

4.3. Rotation Matrix

Rotation is crucial to create the radial symmetry of mandalas. After a base petal is created, it must be replicated multiple times around a circle. For this, we compute an angle θ based on the total number of sectors.

$\boldsymbol{\theta = (2\pi \times i) / \mathrm{num\_sectors}}$

, where *i* is the sector index.

The rotation matrix is:

$\boldsymbol{R(\theta) = [\cos(\theta)\ -\sin(\theta);\ \sin(\theta)\ \cos(\theta)]}$

This matrix is applied to every point on the petal to rotate it into its sector position. By repeating this for all sectors, the complete radial design is formed. The elegance of the rotation matrix lies in how it maintains the shape’s size and angles while changing its orientation.

4.4. Reflection Matrices

Reflection is a linear transformation that creates a mirror image of a shape across a specified axis. In our project, we used reflection across the Y-axis to enhance the symmetry of the mandala design.

The reflection matrix over the Y-axis is:

$\boldsymbol{\mathrm{Ref} = [-1\ 0;\ 0\ 1]}$

In context, once each petal has been rotated to its angular position, we apply this reflection matrix to mirror the petal across the Y-axis. This helps create a balanced, flower-like appearance, where each petal has a corresponding mirror on the opposite side. The reflection does not alter the shape but simply flips it over the central axis, showcasing the power of linear algebra in graphical transformations.

4.5. Shear Transformation

Shear transformation slants the shape of an object along the X or Y axis, effectively distorting it without changing its area. It is defined by the shear matrix:

$\boldsymbol{S = [\ 1\ k_x;\ k_y\ 1]}$

where k_x and k_y are shear factors. If both are zero or the toggle is off, the matrix becomes an identity matrix (no shear).

This shearing adds a creative distortion to the petals, making them appear slanted or dynamically stretched. The result is an asymmetrical yet artistically pleasing distortion that contributes to the uniqueness of each mandala.

4.6. Scaling Transformation

Scaling modifies the size of each petal depending on the layer it belongs to. The outermost layer has the largest petals, while inner layers are progressively smaller, creating a concentric, nested look typical of mandalas.

This is achieved by multiplying each point of the petal by a scalar scale factor:

$\mathbf{scale = base\_scale \times (1 - 0.2 \times (\mathit{l} - 1))}$,

where l is the current layer number.

This reduction provides depth and complexity to the final design while preserving the petal's shape and symmetry.

4.7. Composite Transformations

Multiple transformations are applied sequentially to each petal. For example, a petal might be sheared, then rotated, then reflected. In linear algebra, this is represented by combining transformation matrices using matrix multiplication.

If T_1, T_2, and T_3 are individual transformations, the combined transformation is:

$\mathbf{T = T_3 \times T_2 \times T_1}$

In our project, composite transformations are used when we apply shear, scaling, rotation, and reflection together on the base shape using matrix multiplication before plotting the final mandala design.

4.8. Linear Independence and Rank

Each transformation matrix used (rotation, reflection, etc.) is linearly independent, meaning it introduces a unique effect. The rank of these matrices is 2, as they fully span 2D space and preserve all degrees of freedom. Understanding these properties ensures we do not apply degenerating transformations that collapse the geometry or reduce dimensionality.

4.9. Eigenvalues and Eigenvectors (Insight)

Although not directly used in petal generation, eigenvalues and eigenvectors explain how shapes scale or align under a transformation. For instance, in a shear matrix, the eigenvectors reveal directions that remain unchanged in orientation. This concept could be used in future enhancements like animating transformations or detecting symmetry axes dynamically.

4.10. Dot Product and Angle Between Vectors (Insight)

To understand symmetry and spacing between petals, the dot product can be used to calculate the angle between two vectors. Given vectors u and v:

$$\mathbf{u \cdot v = ||u|| \cdot ||v|| \cdot \cos(\theta) \Rightarrow \theta = \cos^{-1}\{(u \cdot v) / (||u|| \cdot ||v||)\}}$$

While our design primarily uses fixed angles (based on the number of sectors), this mathematical insight helped validate whether evenly spaced petal placements were geometrically accurate or not.

5. MATLAB Concepts Used

5.1. MATLAB Environment Overview

MATLAB (short for *MATrix LABoratory*) is a high-level language and environment designed primarily for matrix computations, algorithm development, data visualization, and GUI creation. Our project heavily relied on MATLAB’s built-in support for matrix manipulation, graphics, and its GUI development tools.

5.2. Matrix Handling and Operations in MATLAB

Since mandala generation is fundamentally based on matrix operations, MATLAB served as the ideal platform. Points defining the petals were stored as 2D arrays (matrices), and all transformations - including rotation, reflection, shear, and scaling - were applied using matrix multiplications.

Example of storing a petal as a set of vectors:

points = [x_values; y_values]; % 2 x N matrix

Applying a transformation:

transformed_points = transformation_matrix * points;

These simple, readable syntax patterns helped ensure the code was efficient and aligned well with the underlying linear algebra principles.

5.3. Plotting and Visualization Basics

MATLAB’s powerful plotting capabilities were used to render the mandala patterns in real time. We used the ‘plot()’ and ‘fill()’ functions to draw each petal and applied ‘axis equal’ to maintain the aspect ratio for accurate geometric display. Layers and sectors were visualized using ‘hold on’ and ‘axis off’ to remove unnecessary axes for artistic clarity.

Example:

```
fill (transformed_points(1, :), transformed_points(2, :), petal_color);

hold on;

axis equal;

axis off;
```

This ensured a clean, symmetric visual output, ideal for intricate designs like mandalas.

5.4. GUI Programming Concepts: Callbacks, Handles, and Properties

The graphical user interface (GUI) was developed using MATLAB's feature of uicontrol and figure. This environment allows GUI elements (buttons, sliders, toggles, etc.) to be linked with code through callbacks — functions triggered by user actions. Each UI element is represented as an object with properties (like 'Value', 'Text', etc.), and handles refer to these objects in the code.

Example callback function for a toggle switch:

```
function ShearToggleSwitchValueChanged(app, event)

app.shearEnabled = app.ShearToggleSwitch.Value;

end
```

This model-view-controller approach makes the GUI highly interactive and responsive to user input.

5.5. Using Programmatic GUI in MATLAB

We chose to create the GUI in MATLAB using code (with uicontrol and figure) instead of using tools like GUIDE or App Designer. This way is more flexible, lighter, and easier to change without needing a separate layout file. It also gave us full control over where buttons and other parts go and how they behave, using short and simple functions. This made our code shorter and easier to manage.

Key features we used:

- Dropdowns and input boxes to select base shape types and parameters like sectors, sides, or curve values.
- Checkboxes to toggle features such as shear transformation and eigenvector display.
- Sliders to adjust the base scaling factor dynamically.
- A background colour selector with real-time preview.

The GUI was structured functionally, with a main callback (generateMandala) handling shape generation, transformation (scaling, shear), and layered rendering using polar and Cartesian math.

5.6. Specific MATLAB Functions/Commands Used

Throughout the development, we made use of various MATLAB functions that enhanced clarity and performance:

- linspace (): For generating smooth parametric curves.
- sin (), cos (): For defining parametric shapes and rotation matrices.
- plot (), fill (): For drawing petals and shapes.
- axis equal, axis off: For clean visualization.
- uistyle (), uicontrol (), uifigure (): For custom GUI elements (where needed).
- str2double (), sprintf (), num2str (): For debugging and display.

The combination of these tools enabled both aesthetic control and precise mathematical manipulation.

6. GUI Implementation

The GUI has been designed for accessibility and customization.

- *Dropdowns* for selecting base shapes and background colours.
- *Edit boxes* for number of sectors/layers, k-value for rose curves.
- *Sliders* and *checkboxes* to enable shear and scaling.
- *Buttons* for generating mandala and saving image.

Each control uses MATLAB's 'uicontrol()' function. For example, the line:

shearCheckbox = uicontrol (f, 'Style', 'checkbox', 'String', 'Apply Shear Transformation');

...adds the checkbox to toggle shear transformation.

7. Output Visualization

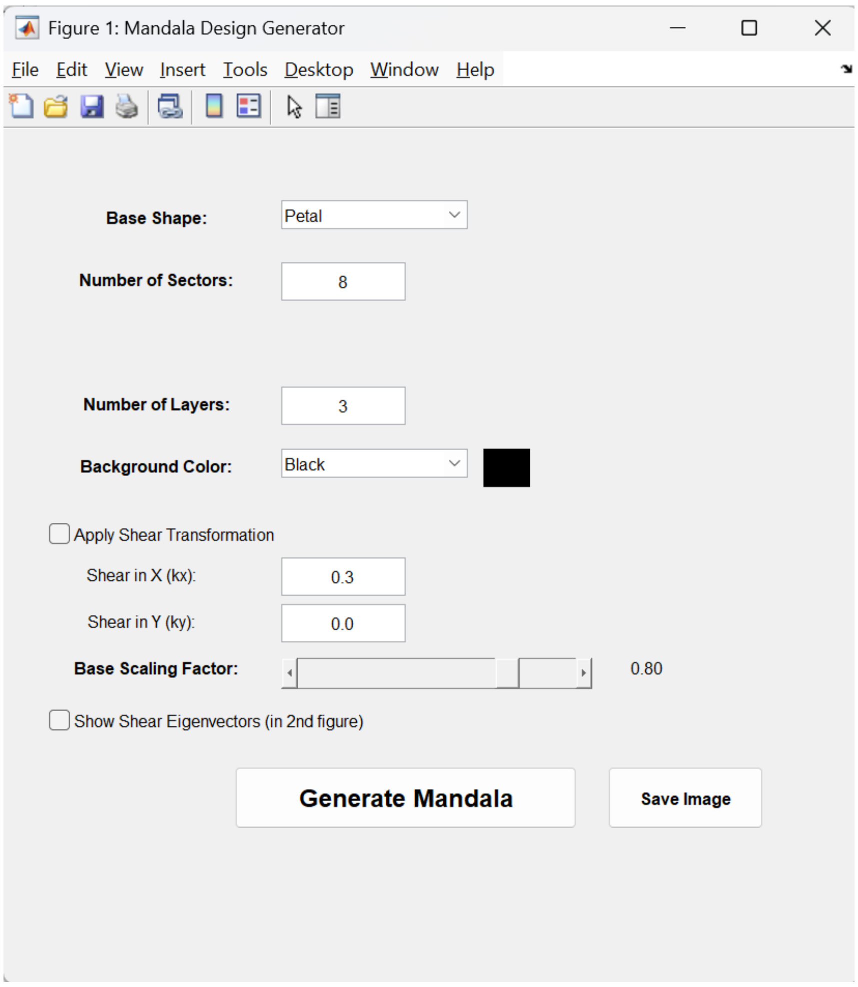
Figure 1: Mandala Design Generator

File Edit View Insert Tools Desktop Window Help

Base Shape: Petal

Number of Sectors: 8

Number of Layers: 3

Background Color: Black

☐ Apply Shear Transformation

Shear in X (kx): 0.3

Shear in Y (ky): 0.0

Base Scaling Factor: 0.80

☐ Show Shear Eigenvectors (in 2nd figure)

Generate Mandala

Save Image

Figure 1: The code when run generates a GUI based user interface with some default values.

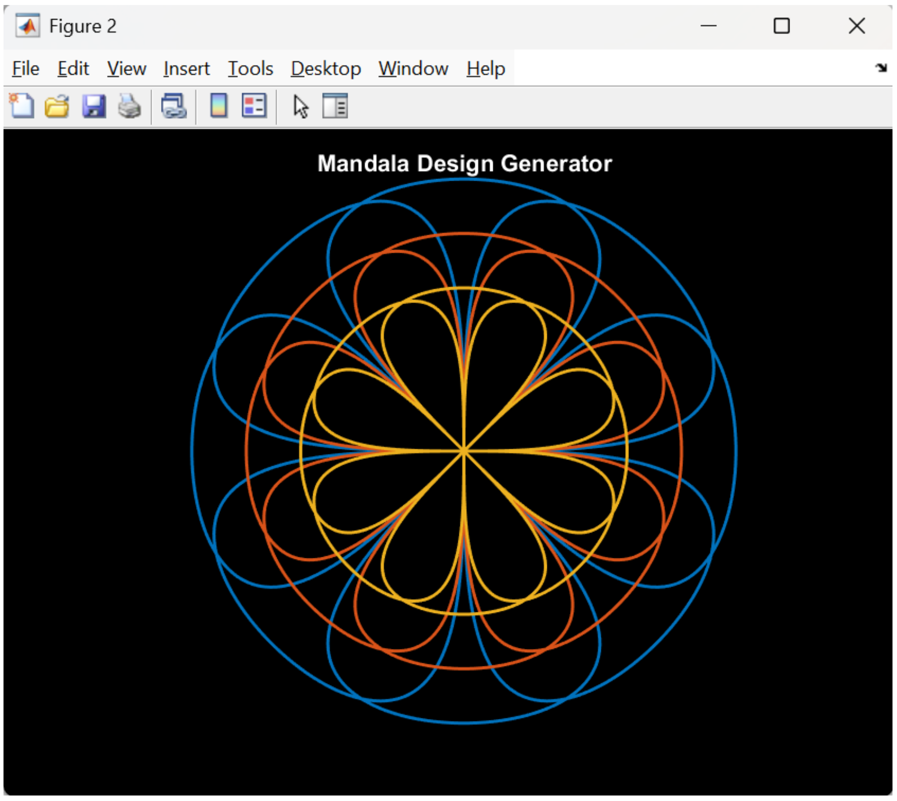

Figure 2: Default Input Mandala Generation

7.1. Petal-Based Mandala

This shape is constructed using sine and cosine variations with the equation:

$x(t) = \sin(t)$ & $y(t) = \cos(t) \times \sin(t)$ ………, where $t \in [0, \pi]$

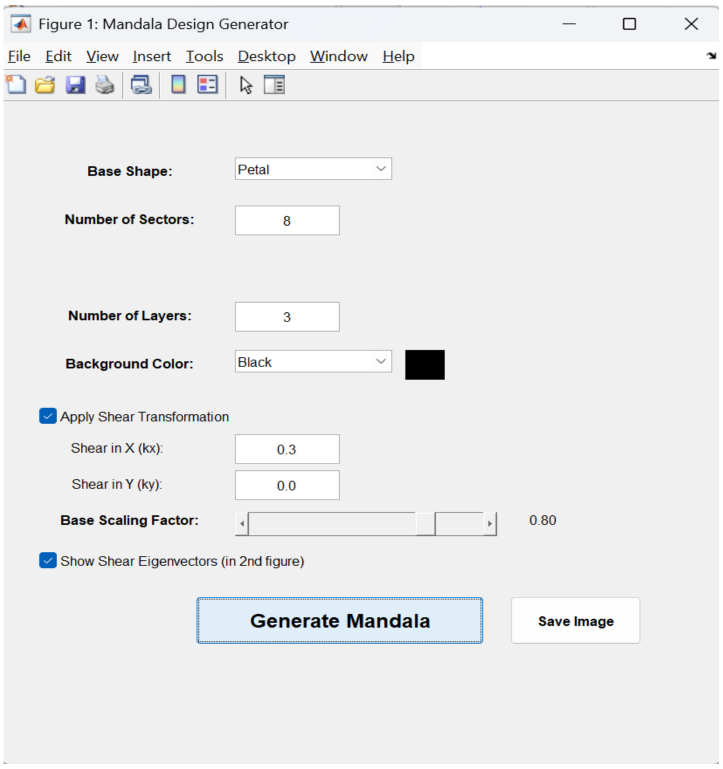
Figure 1: Mandala Design Generator
File Edit View Insert Tools Desktop Window Help
Generate Mandala
Save Image

Base Shape: Petal

Number of Sectors: 8

Number of Layers: 3

Background Color: Black

[✓] Apply Shear Transformation

Shear in X (kx): 0.3

Shear in Y (ky): 0.0

Base Scaling Factor: 0.80

[✓] Show Shear Eigenvectors (in 2nd figure)

Figure 3: GUI Panel for Petal Based Mandala with Shear Transformation

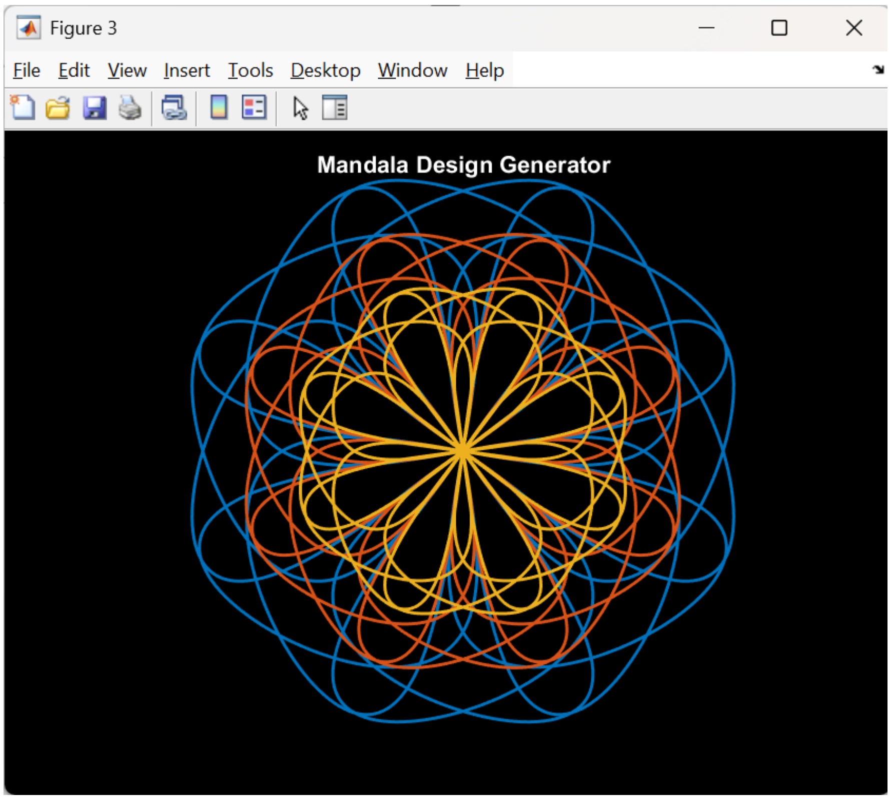

Figure 4: Petal-Based Mandala Design with Shear Transformation

Only x component has a value of 0.3 hence a horizontal vector parallel to x axis.

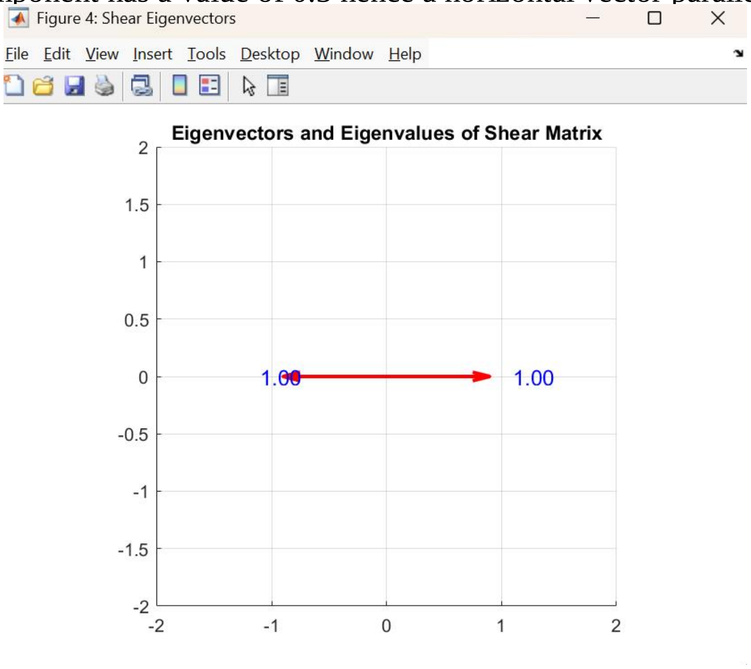

Figure 5: Visualizing Eigenvectors for the above Shear Transformation Matrix

7.2. Polygon-Based Mandala

Polygons are generated using vertex plotting, diagonals add aesthetic layering to design. The equation used to generate the pattern is:

x(θ) = cos(θ) & y(θ) = sin(θ)..., where θ ∈ [0, 2π] and number of sides = user input

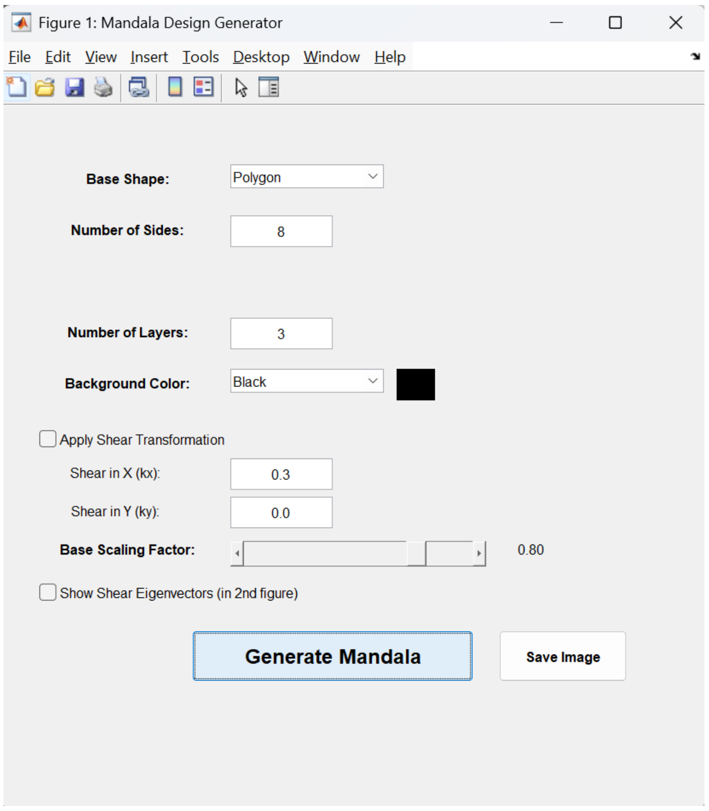
Figure 1: Mandala Design Generator
File Edit View Insert Tools Desktop Window Help
Generate Mandala
Save Image

Base Shape:	Polygon
Number of Sides:	8
Number of Layers:	3
Background Color:	Black ⬛
☐ Apply Shear Transformation	
Shear in X (kx):	0.3
Shear in Y (ky):	0.0
Base Scaling Factor:	0.80
☐ Show Shear Eigenvectors (in 2nd figure)	

Figure 6: GUI Panel for Polygon-Based Mandala Design

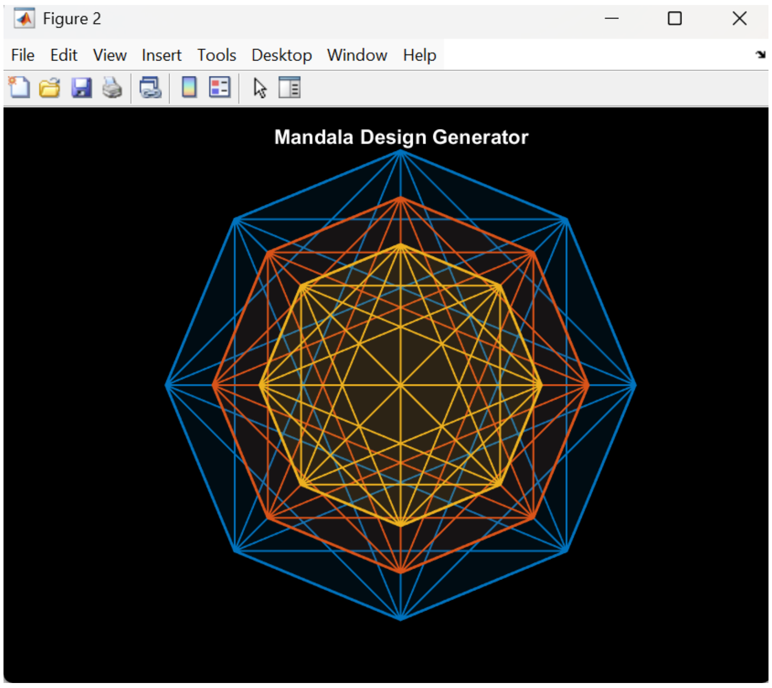

Figure 7: Polygon-Based Mandala Design

7.3. Rose Curve-Based Mandala

The base shape (2 petals opposite of each other) is created by the following equations:

r(t) = cos (k × t) & x(t) = r(t) × cos(t) = cos (k × t) × cos(t)

y(t) = r(t) × sin(t) = cos(k × t) × sin(t)…, where t ∈ [0, 2π / k] and k = number of petals

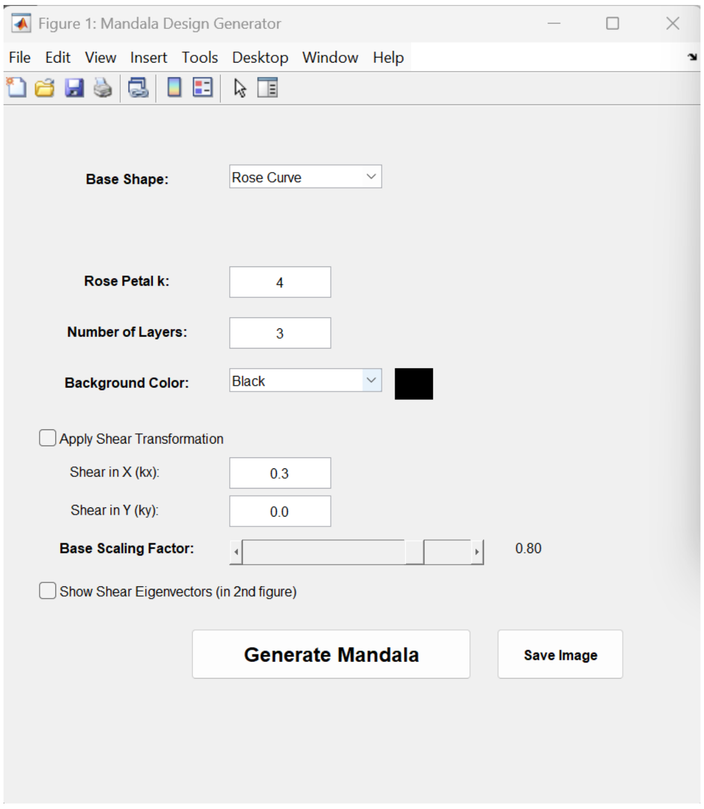
Figure 1: Mandala Design Generator
File Edit View Insert Tools Desktop Window Help
Generate Mandala
Save Image

Base Shape:	Rose Curve
Rose Petal k:	4
Number of Layers:	3
Background Color:	Black
☐ Apply Shear Transformation	
Shear in X (kx):	0.3
Shear in Y (ky):	0.0
Base Scaling Factor:	0.80
☐ Show Shear Eigenvectors (in 2nd figure)	

Figure 8: GUI Panel for Rose Curve-Based Mandala Design

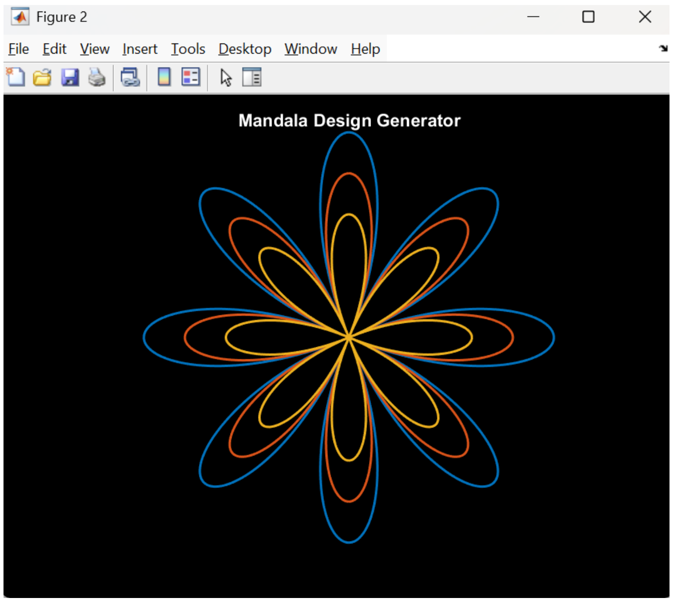

Figure 9: Rose Curve-Based Mandala Design

8. Appendix

8.1. Complete MATLAB code

```
function mandala()
% Create GUI Figure
f = figure('Name', 'Mandala Design Generator', 'Position', [300 200 550 550]);

% ================= Input Fields (GUI Setup) ===================

% Base Shape Dropdown
uicontrol(f, 'Style', 'text', 'Position', [30 480 140 20], 'String', 'Base Shape:', ...
    'FontWeight', 'bold');
shapeDropdown = uicontrol(f, 'Style', 'popupmenu', ...
    'Position', [180 480 120 25], ...
    'String', {'Petal', 'Polygon', 'Rose Curve'}, ...
    'Callback', @updateShapeInputs);

% Number of Sectors or Polygon Sides
label1 = uicontrol(f, 'Style', 'text', 'Position', [30 440 140 20], 'String', 'Number
of Sectors:', ...
    'FontWeight', 'bold');
inputBox1 = uicontrol(f, 'Style', 'edit', 'Position', [180 440 80 25], 'String', '8');

% For Rose Curve - k value
labelK = uicontrol(f, 'Style', 'text', 'Position', [30 400 140 20], 'String', 'Rose
Petal k:', ...
    'FontWeight', 'bold', 'Visible', 'off');
kBox = uicontrol(f, 'Style', 'edit', 'Position', [180 400 80 25], 'String', '4',
'Visible', 'off');

% Number of Layers
```

```
uicontrol(f, 'Style', 'text', 'Position', [30 360 140 20], 'String', 'Number of
Layers:', ...
    'FontWeight', 'bold');
layerBox = uicontrol(f, 'Style', 'edit', 'Position', [180 360 80 25], 'String', '3');

% Background color dropdown with preview
uicontrol(f, 'Style', 'text', 'Position', [30 320 140 20], 'String', 'Background
Color:', ...
    'FontWeight', 'bold');
bgColors = {'Black', 'Red', 'Green', 'Blue', 'Yellow', 'Cyan', 'Magenta', 'White'};
bgColorMap = containers.Map(bgColors, {'k','r','g','b','y','c','m','w'});

bgDropdown = uicontrol(f, 'Style', 'popupmenu', ...
    'Position', [180 320 120 25], ...
    'String', bgColors, ...
    'Callback', @updatePreview);

bgPreview = uicontrol(f, 'Style', 'text', ...
    'Position', [310 320 30 25], ...
    'BackgroundColor', 'k');

    function updatePreview(src, ~)
        colorName = bgColors{src.Value};
        set(bgPreview, 'BackgroundColor', bgColorMap(colorName));
    end

% Shear
shearCheckbox = uicontrol(f, 'Style', 'checkbox', ...
    'String', 'Apply Shear Transformation', ...
    'Position', [30 280 200 20]);

uicontrol(f, 'Style', 'text', 'Position', [30 250 120 20], 'String', 'Shear in X
(kx):');
kxBox = uicontrol(f, 'Style', 'edit', 'Position', [180 250 80 25], 'String', '0.3');

uicontrol(f, 'Style', 'text', 'Position', [30 220 120 20], 'String', 'Shear in Y
(ky):');
kyBox = uicontrol(f, 'Style', 'edit', 'Position', [180 220 80 25], 'String', '0.0');

% Scaling Slider
uicontrol(f, 'Style', 'text', 'Position', [30 190 140 20], 'String', 'Base Scaling
Factor:', ...
    'FontWeight', 'bold');
scaleSlider = uicontrol(f, 'Style', 'slider', ...
    'Min', 0.1, 'Max', 1.0, 'Value', 0.8, ...
    'Position', [180 190 200 20]);
scaleText = uicontrol(f, 'Style', 'text', ...
    'Position', [390 190 50 20], 'String', '0.80');
scaleSlider.Callback = @(src, ~) set(scaleText, 'String', num2str(get(src, 'Value'),
'%.2f'));

% Eigenvectors Checkbox
```

```
eigenCheckbox = uicontrol(f, 'Style', 'checkbox', ...
    'String', 'Show Shear Eigenvectors (in 2nd figure)', ...
    'Position', [30 160 300 20]);

% Generate Button
uicontrol(f, 'Style', 'pushbutton', 'String', 'Generate Mandala', ...
    'FontWeight', 'bold', 'FontSize', 12, ...
    'Position', [150 100 220 40], ...
    'Callback', @generateMandala);

% Save Button
uicontrol(f, 'Style', 'pushbutton', 'String', 'Save Image', ...
    'FontWeight', 'bold', ...
    'Position', [390 100 100 40], ...
    'Callback', @(~,~) saveas(gcf, 'mandala.png'));

% === Callbacks to adjust input fields based on shape ===
    function updateShapeInputs(src, ~)
        val = src.Value;
        if val == 1  % Petal
            label1.String = 'Number of Sectors:';
            label1.Visible = 'on';
            inputBox1.Visible = 'on';
            labelK.Visible = 'off';
            kBox.Visible = 'off';
        elseif val == 2  % Polygon
            label1.String = 'Number of Sides:';
            label1.Visible = 'on';
            inputBox1.Visible = 'on';
            labelK.Visible = 'off';
            kBox.Visible = 'off';
        elseif val == 3  % Rose Curve
            label1.Visible = 'off';
            inputBox1.Visible = 'off';
            labelK.Visible = 'on';
            kBox.Visible = 'on';
        end
    end

% ============== Callback to Generate Mandala ===============
    function generateMandala(~, ~)
        shapeType = shapeDropdown.Value;
        num_layers = str2double(get(layerBox, 'String'));
        bg_color = bgColorMap(bgColors{get(bgDropdown, 'Value')});
        apply_shear = get(shearCheckbox, 'Value');
        kx = str2double(get(kxBox, 'String'));
        ky = str2double(get(kyBox, 'String'));
        base_scale = get(scaleSlider, 'Value');
        show_eigen = get(eigenCheckbox, 'Value');

        % Shear Matrix
        if apply_shear
```

```
            S = [1 kx; ky 1];
        else
            S = eye(2);
        End

        % Setup Mandala Plot
        figure;
        axis equal; hold on; axis off;
        set(gcf, 'Color', bg_color);
        title('Mandala Design Generator', 'Color', 'w');

        colors = lines(num_layers);
        switch shapeType
            case 1 % Petal
                num_sectors = str2double(get(inputBox1, 'String'));
                t = linspace(0, pi, 100);
                base_x = sin(t);
                base_y = cos(t) .* sin(t);
                base_shape = [base_x; base_y];

                theta_step = 2 * pi / num_sectors;
                for layer = 1:num_layers
                    scale = base_scale * (1 - (layer - 1) * 0.2);
                    shape = S * (base_shape * scale);
                    for i = 0:num_sectors - 1
                        theta = i * theta_step;
                        R = [cos(theta), -sin(theta); sin(theta), cos(theta)];
                        rotated = R * shape;
                        plot(rotated(1, :), rotated(2, :), 'Color', colors(layer, :),
'LineWidth', 1.5);

                        Ref = [-1 0; 0 1];
                        reflected = R * Ref * shape;
                        plot(reflected(1, :), reflected(2, :), 'Color', colors(layer,
:), 'LineWidth', 1.5);
                    end
                end
            case 2 % Polygon
                n_sides = str2double(get(inputBox1, 'String'));
                theta = linspace(0, 2*pi, n_sides+1);
                base_shape = [cos(theta); sin(theta)];
                for layer = 1:num_layers
                    scale = base_scale * (1 - (layer - 1) * 0.2);
                    shape = S * (base_shape * scale);
                    fill(shape(1,:), shape(2,:), colors(layer,:), 'FaceAlpha', 0.1,
'EdgeColor', colors(layer,:), 'LineWidth', 1.5);

                    % Draw diagonals
                    for i = 1:n_sides
                        for j = i+2:n_sides
                            if j ~= mod(i-2, n_sides)+1
```

```
                                plot([shape(1,i), shape(1,j)], [shape(2,i), 
shape(2,j)], ...
                                    'Color', colors(layer,:), 'LineStyle', '-', 
'LineWidth', 1.0);
                            end
                        end
                    end
                end

            case 3 % Rose Curve
                k = str2double(get(kBox, 'String'));
                % Single petal range
                t = linspace(0, 2*pi/k, 200);
                r = cos(k * t);
                base_x = r .* cos(t);
                base_y = r .* sin(t);
                base_shape = [base_x; base_y];

                theta_step = 2 * pi / k;
                for layer = 1:num_layers
                    scale = base_scale * (1 - (layer - 1) * 0.2);
                    shape = S * (base_shape * scale);

                    for i = 0:k-1
                        theta = i * theta_step;
                        R = [cos(theta), -sin(theta); sin(theta), cos(theta)];
                        rotated = R * shape;
                        plot(rotated(1, :), rotated(2, :), 'Color', colors(layer, :), 
'LineWidth', 1.5);
                    end
                end
        end
        % To show shear eigenvectors in separate figure
        if show_eigen && apply_shear
            figure('Name', 'Shear Eigenvectors');
            axis equal; hold on; grid on;
            set(gcf, 'Color', 'w');
            title('Eigenvectors and Eigenvalues of Shear Matrix');

            [V, D] = eig(S);
            for i = 1:2
                vec = V(:,i);
                lambda = D(i,i);
                quiver(0, 0, vec(1), vec(2), 'r', 'LineWidth', 2, 'MaxHeadSize', 0.5);
                text(vec(1)*1.1, vec(2)*1.1, sprintf('%.2f', lambda), 'FontSize', 12, 
'Color', 'b');
            end
            xlim([-2 2]); ylim([-2 2]);
        end
    end
updateShapeInputs(shapeDropdown, []);
end
```

9. Acknowledgment

We would like to express our sincere gratitude to all those who contributed to the successful completion of this project. We are grateful for the guidance provided by our mentors and professors and appreciate the support provided by our fellow batchmates and collaborators for their active involvement.

10. References

10.1. Linear Algebra and Its Applications (5th Edition), By David C. Lay, Steven R. Lay, Judi J. McDonald [Book]
10.2. Introduction to Linear Algebra (5th Edition), By Gilbert Strang. [Book]
10.3. The MathWorks, Inc. (2024). MATLAB Documentation.
10.3.1. https://www.mathworks.com/help/
10.4. MIT OpenCourseWare, 18.06 Linear Algebra, By Gilbert Strang.
10.4.1. https://ocw.mit.edu/courses/18-06sc-linear-algebra-fall-2011/
10.5. MATLAB GUI w/o GUIDE: figure, uicontrol, push buttons, edit boxes
10.5.1. https://www.youtube.com/watch?v=THGC0Hd3K9g