Deccan Education Society’s

**FERGUSSON COLLEGE (AUTONOMOUS), PUNE**

--|| DEPARTMENT OF PHYSICS ||--

PROJECT REPORT ON

**3D Interactive Models in Physics**

**using p5.js**

Submitted by

**Vyas Ashish Mokal**

Under the Guidance of

**Dr. Raka V. Dhabade**

DEPARTMENT OF PHYSICS

Fergusson College (Autonomous) Pune

Deccan Education Society’s

FERGUSSON COLLEGE (AUTONOMOUS)

DEPARTMENT OF PHYSICS

**CERTIFICATE**

This is to certify that Mr. Vyas Ashish Mokal, Roll No. 212025 of T.Y.B.Sc Physics class of this department has satisfactorily completed the project entitled*:*

**“3D Interactive Models in Physics using P5.js”**

As partial completion of B.Sc. (Physics) degree University during the year 2022-23.

**External Examiner**

**Dr. Raka V. Dabhade  
Head of Department  
Fergusson College, Pune**

**Date:**

**Dr. Raka V. Dabhade  
Guide, Department of Physics  
Fergusson College, Pune**

**Internal Examiner**

**Acknowledgement**

*I thank to my project guide, Dr. Raka Dabhade, for her unwavering belief in the potential of this project and her invaluable insights and guidance throughout its development.*

*I am thankful to my PHY3502 “Solid States of Physics” instructor Dr. Swati Gaikwad for her kind appreciation of the initial Models which I made. It fueled my interest to delve deeper into the work.*

*I am deeply grateful to the YouTube channel "3blue1brown" by Grant Sanderson for their exceptional animated videos on mathematics and physics. These insightful resources served as the initial source of motivation for this project.*

*I would also like to express my gratitude to the YouTube channel "The Coding Train" by Daniel Schiffman for his outstanding tutorials on Processing software and p5.js. Their engaging videos provided essential knowledge and guidance, significantly contributing to the development of this project.*

*Furthermore, I am grateful to the creator and contributors of the GitHub repository "js-expression-eval," which converts the string into JavaScript functions. It played a crucial role in the vector field plotter, enabling to provide vector field components as user-defined functions.*

*Lastly, I would thank to all those who have contributed their time, expertise, and unwavering support, directly or indirectly, I extend my sincere gratitude and deepest appreciation.*

**Index**

|  |  |
| --- | --- |
| **Chapter 1 Introduction** | **5** |
| **Chapter 2 Theoretical Background**   1. What is p5js and WEBGL 7 2. Creating working Environment 7 3. Basic objects in P5js 8 4. Events 10 5. Camera, View, Lighting and Material 11 6. Solid State Physics Models 15 7. Fourier Series 16 8. Vector Field Plotter 18 9. Lissajous Figures 21 | **7** |
| **Chapter 3 Program** | **23** |
| **Chapter 4 Results and Discussion** |  |
| **Chapter 5 Conclusion and Future Scope** |  |
| **Chapter 6 References** |  |

**1. Introduction**

This project is born out of a passion for combining the power of coding with the beauty of physics. Inspired by the enlightening approach of 3Blue1Brown [[1]](https://www.youtube.com/@3blue1brown) and the capabilities of p5.js, this endeavor seeks to bridge the gap between static 3D images and a deep understanding of complex structures.

In physics, the power of visualization cannot be overstated. Traditional static images often fall short in conveying the intricacies of crystal structures, vector fields, 3D functions and other concepts. Recognizing this limitation during a solid-state physics lecture, I embarked on a coding journey initially for the sheer joy of learning and exploring.

Having previously created a simple cubic model using Processing, the project gradually evolved as I discovered the versatility of p5.js, a JavaScript library designed for drawing and animations. What made it even more appealing was its compatibility with any web browser, eliminating the need for specific software installations.

The incorporation of WEBGL in p5.js opened up exciting possibilities for creating intricate 3D structures. The project offers a platform for interactive 3D models with a dynamic experience, allowing users to explore and understand complex physical phenomena effortlessly.

The project currently encompasses an array of interactive models, Solid State Models, Fourier Series, Vector Fields Plotter and Lissajous Figures. Solid State Models are coded in Processing Software while the others uses WEBGL incorporated in p5js.

Solid-state models span various unit cells such as Simple Cubic, Face-Centered Cubic, Body-Centered Cubic, Hexagonal, Diamond Cubic, alongside representations of voids, NaCl, and CsCl structures. Additionally, the project features a Miller Indices plotter, empowering users to input Miller indices (hkl), thereby generating corresponding plane plots for visualization and analysis.

Fourier series visualization incorporates square and saw wave representations, providing users and learners with a tangible understanding of individual terms within the Fourier series expansion and discerning why sines and cosines of increasing frequency are integral components of the expansion, symbolizing the radii of circles and frequencies of rotation within the model.

The Vector Field Plotter stands out as a distinctive model within our project, offering dynamic visualization capabilities for both 2D and 3D vector fields. Users have the flexibility to define their own vector fields, empowering them to explore and analyze various scenarios. A noteworthy feature of this model is the representation of vector magnitudes through color, providing intuitive insights into the field's characteristics. Coupled with other features designed to enhance user understanding.

Finally, project includes Lissajous figures, elegantly illustrating how the superposition of perpendicular simple harmonic motions generates intricate patterns. Users have the freedom to select their preferred frequencies and phase differences, enabling them to customize their exploration.

The project is open to all and one can find it on my Git Hub Repository <https://vyasmokalzz.github.io/Physics-Models/>. Overall User Interface is kept very convenient and easy to handle. The site currently only works on desktop browsers and mobile support is not incorporated. The aim is to visualize these concepts and provide an engaging platform for learning and exploration.

**2. Theoretical Background**

**2.1 What is p5js and WEBGL**

p5.js [[2]](https://p5js.org/) is a free and open-source JavaScript library for creative coding, with a focus on making coding accessible and inclusive for artists, designers, educators, beginners, and anyone else.

WebGL [[3]](https://p5js.org/learn/getting-started-in-webgl-coords-and-transform.html), or Web Graphics Library, is a JavaScript API (Application Programming Interface) that enables the rendering of 3D and 2D graphics within web browsers. It provides a bridge between the browser and the computer's GPU (Graphics Processing Unit), allowing for hardware-accelerated graphics rendering.

A basic p5.js program incorporating WEBGL looks like this:

function setup() {

    createCanvas(windowWidth, windowHeight, WEBGL);

    describe('a red box on a white background');

  }

  function draw(){

    background(255);

    fill(255,0,0);

    box();

  }

One of the most fundamental differences between working in 2D and working in 3D is the most obvious: there is one more dimension to work with. In addition to the horizontal and vertical position (x and y axes) of an element in our drawing, 3D adds depth, the z-axis.

When drawing in 2D, the point (0,0) is located at the top left corner of the screen. In WebGL mode, the origin of the sketch (0,0,0) is located in the middle of the screen. By default, the x-axis goes left-to-right, y-axis goes up-to-down, and the z-axis goes from further-to-closer.

**2.2 Creating working Environment**

Throughout the project I have used Visual Studio Code [[4]](https://code.visualstudio.com/) (VS Code) as code editor. To incorporate p5js into webpage I used VS Code extension p5.vscode [[5]](https://marketplace.visualstudio.com/items?itemName=samplavigne.p5-vscode). It sets up the entire coding environment using Command Pallet, giving libraries, CSS, JavaScript and HTML files. The sketch can be edited in sketch.js file as shown below.

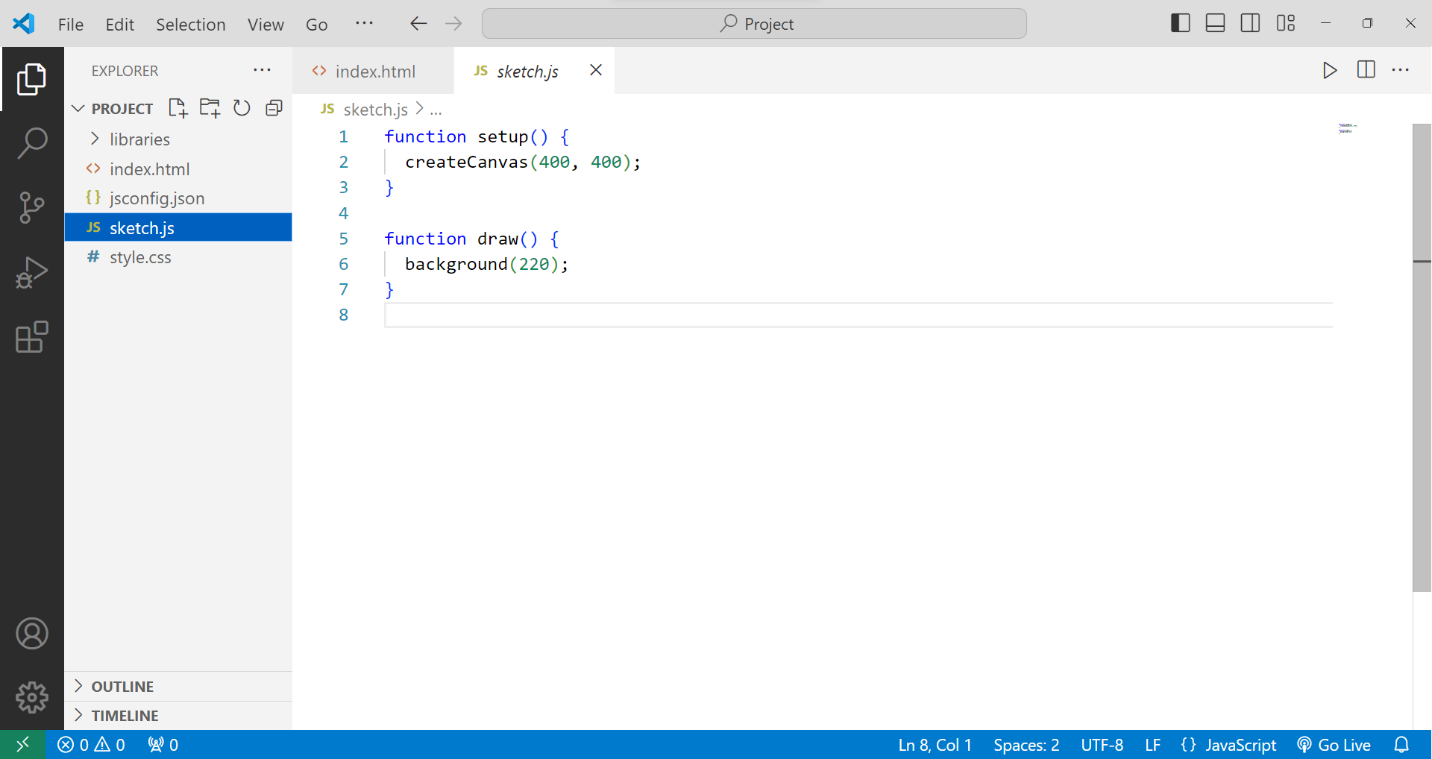


Image 1: Basic Working Environment for a p5js Model

**2.3 Basic Objects in p5js**

In p5js we can create variety of shapes which includes arc, ellipse, circle, line, point, quadrilateral, rectangle, square, triangle and their corresponding functions are arc(), ellipse(), circle(), line(), point(), quad(), rect(), square(), triangle() respectively. Consider the following example

function setup() {

  createCanvas(400, 300);

}

function draw() {

  background(220);

  stroke(0);

  fill(255,0,0);  //specify color in RGB

  rect(50,50,150,100);  // specify (x1,y1) and (x2,y2) corner coordinates

  fill(0,0,255);

  circle(300,150,100);  // specify center and radius

}

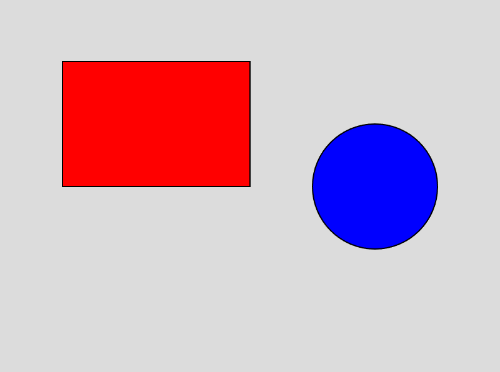
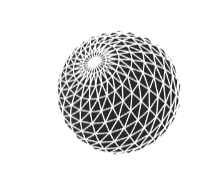
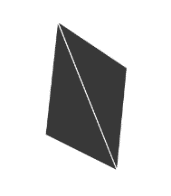
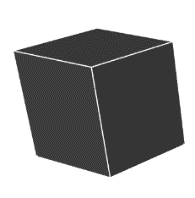
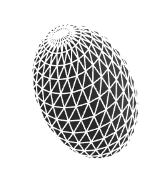


Figure 1: Output of the Previous Code

Similarly, we can create 3D Objects incorporating WEBGL into the project. Few supported 3D shapes are box, plane, sphere, ellipsoid, cone, cylinder and torus with their functions being box(), plane(), sphere(), ellipsoid(), cone(), cylinder() and torus() respectively.

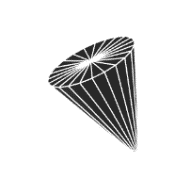
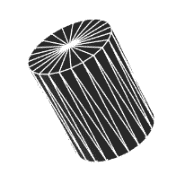
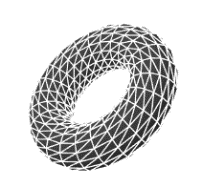


Ellipsoid

Sphere

Plane

Box



Torus

Cylinder

Cone

Figure 2: Objects in p5js

One more example of the same:

function setup() {

  createCanvas(400, 400, WEBGL);

}

function draw() {

  background(220);

  stroke(0);

  noFill();

  rotateX(PI/3);

  rotateY(PI/3);

  box(100);

}

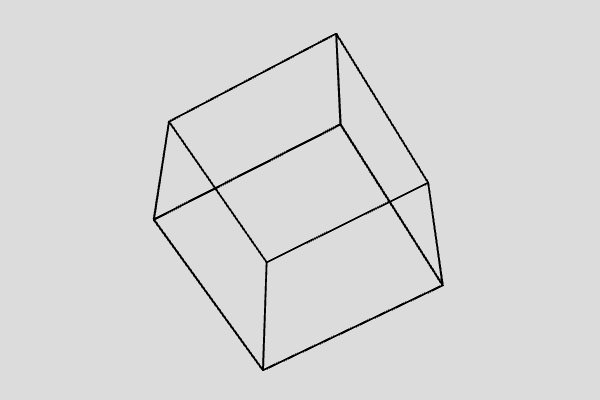


Figure 3: A 3D Cube in p5js

**2.4 Events**

A category of functions called events alter the normal flow of a program when an action such as a key press or mouse movement takes place. An event is a polite interruption of the normal flow of a program.

Events like keypresses and mouse press allow us to interact with the computer, hence would facilitate to make our models more interactive to the user, allowing to change the data and orientation according to user’s requirement.

P5js allows us to manipulate a number of mouse and keyboard event.  
Mouse Events

* mousePressed() - Code inside this block is run one time when a mouse button is pressed
* mouseReleased() - Code inside this block is run one time when a mouse button is released
* mouseClicked() - Code inside this block is run once after a mouse button is pressed and released over the element
* doubleClicked() - Code inside this block is run once after a mouse button is pressed and released over the element twice
* mouseMoved() - Code inside this block is run one time when the mouse is moved
* mouseDragged() - Code inside this block is run one time when the mouse is moved while a mouse button is pressed
* mouseOver() - Code inside this block is run once after every time a mouse moves onto the element.
* mouseOut() - Code inside this block is run once after every time a mouse moves off the element

Keyboard Events

* keyPressed() – Code inside this block is run one time when a mouse button is pressed.
* keyTyped() – Code inside this block is run one time when a key is pressed, but action keys such as Ctrl, Shift, and Alt are ignored. The most recent key pressed will be stored in the key variable.
* keyReleased() – Code inside this block is run one time when any key is released.

**2.5 Camera, View, Lighting and Material**

Other than just geometry creating 3D models requires to take care of Camera, Lighting and materials to make a visually interesting 3D scene.

**2.5.1 Camera**

Camera in p5js give us the sense of space and dimension that we are looking in 3D. In p5.js, the WebGL mode provides us with a perspective camera by default, but we can change this using `perspective()` or `ortho()`.

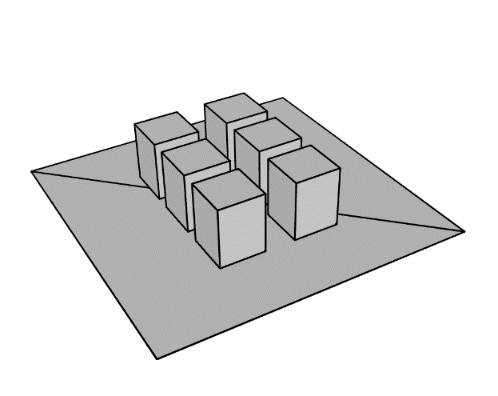
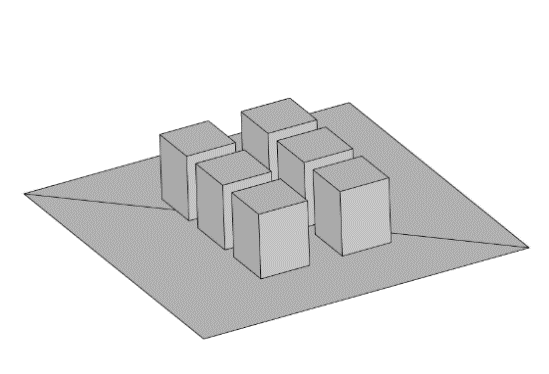


Figure 4 (b): Orthographic Projection

Figure 4 (a): Perspective Projection

A perspective camera skews objects so they appear to get smaller as they get further away, vanishing at a single point in the distance. This is in contrast to an orthographic camera, where the geometry stays the same size as it gets further away and has no vanishing point.

The perspective mode comes with several parameters perspective(fovy, aspect, near far)  
(see fig. below)

* fov: camera frustum vertical field of view, from bottom to top of view as angle (Optional)
* aspect: camera frustrum aspect ratio.
* near: near frustrum plane length.
* far: far frustrum plane length.

an illustration showing the the camera frustum in purple, the near plane represented by a 
        yellow rectangle near the camera, and far plane as a green rectangle on the opposite end of the frustum volume.

Figure 5: Perspective Parameters in p5.js  
Credits: Processing Foundation

I have not used the orthographic projection hence will not go into its details. One can find it in the reference [[8]](https://p5js.org/learn/getting-started-in-webgl-appearance.html).

In order to view the scene from different angles and magnification, p5js has built in orbitControl() function which allows user to freely zoom and position the location of camera. One of the problem with orbitControl is completely free motion, making it hard to keep track of the axes and proper orientation to view the model.

To address the issue and optimize user experience within p5.js 3D scene visualization, a custom function named orbit() is made. Based on spherical polar coordinates [9] (see image 8) impose following conditions on azimuthal angle and zenith angle .

Now instead of changing camera position, camera is kept at one place and the entire scene is rotated by changing and using rotateX() and rotateZ() functions respectively, offset being provided by displacement of the mouse.

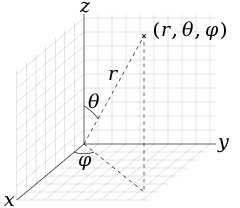


Figure 6: Spherical Polar Coordinates

To zoom the scene Field of View (fov) parameter of perspective() is used where change value in magnification is provided by mouse scroll creating similar effect to that of zoom.

**2.5.2 Lights**

In p5.js, the utilization of lights plays a crucial role in representing the depth of 3D structures effectively. Therefore, incorporating a light source into the program is imperative. Typically, a light source is integrated within the draw() function. p5.js offers various options for implementing lighting techniques within the program.

* ambientLight(): Ambient light makes everything display a little brighter, with no consideration for light position or direction.
* directionalLight(): A directional light shines from one direction, which can be especially useful for communicating depth in a scene, or when a scene needs a 'sun' light. This method accepts a color and direction.
* pointLight(): A point light emits from a single point in all directions, similar to something like a lightbulb. This method accepts a color and a position for the light.
* spotLight(): A spot light emits from a single point in a single direction. This light is cast in a conical shape and it's radius and concentration can be adjusted.

In addition to these types of lighting p5js has one more function lights(). It sets default ambient and directional light in the scene. The default ambient and directional lights come with parameters; directionalLight(128,128,128,0,0,-1) and ambient light(128,128,128). Where first three set of argument represent Red, Green and Blue (RGB) offsets and next three in directional light the direction of the light.

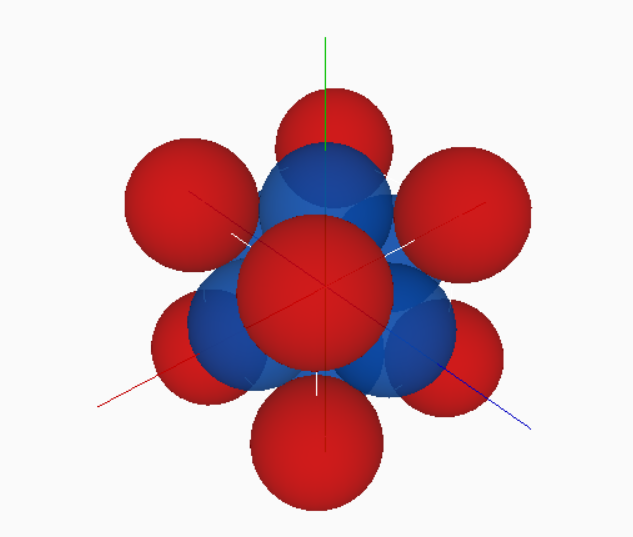
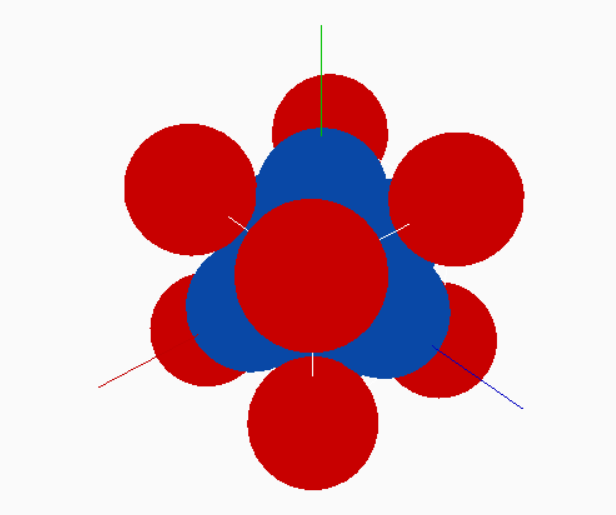
Here is an example of scene with and without lights.

Figure 7 (a): Scene without Lighting

Figure 7 (b): Scene with Lighting

Both Vector Fields Model and Solid State Models use lights() function for lighting in the Program.

**2.5.2 Material**

The appearance of objects can vary significantly depending on their material composition. Different materials influence the way light interacts with the object's surface, as well as the application of color or texture. Objects may exhibit various visual characteristics such as shininess, roughness, or the incorporation of textured patterns, all of which are determined by the properties of the materials used in their construction. In p5.js we have several types of material as listed below.

* **Basic material:** fills the geometry by given color and is not affected by the lightings in the scene.
* **normalMaterial(): .** A normal material is not affected by any light and also does not reflect any light. The surface facing the x-axis becomes red, the surface facing the y-axis becomes green and the surface facing the z-axis becomes blue.
* **ambientMaterial():** It reflects light of a particular color, given as argument to function.
* **specularMaterial():** It is a realistic type of material which reflects light and creates a shining reflecting surface.

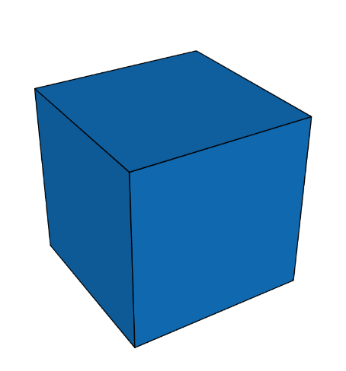
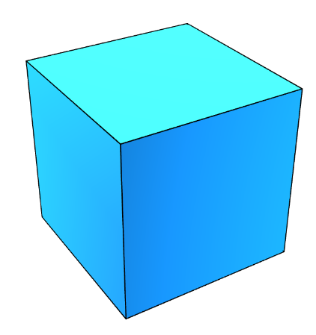
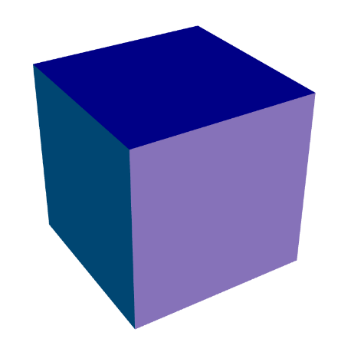
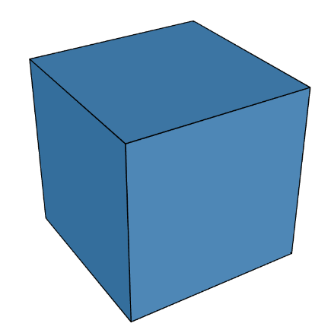


Figure 8(d): Specular Material

Figure 8(c): Ambient Material

Figure 8(b): Normal Material

Figure 8(a): Basic Material

In the program I have used only Basic Material using the fill() function and Ambient Material, as they interact well with normal lights() setup and other materials require complex light settings.

**2.6 Solid State Physics Models**

These models are made using Processing software. It contains various models such as Simple Cubic, Body Centered, Face Centered, Hexagonal, Diamond Cubic, Miller’s Indices, Voids, CsCl and NaCl structures.

The unit cell, void and compound models are simple to make and make use of makeSphere() function to represent an atom at lattice point.

function makeSphere(x, y, z, radius){

    pushMatrix();

    translate(x, y, z);

    sphere(radius);

    popMatrix();

}

Where the transformations are applied only on object(s) within pushMatrix() and popMatrix(). Making other objects independent of these transformations. To plot the given miller Indices we first need to find the intercepts of the corresponding plane. Consider the following figure 9.

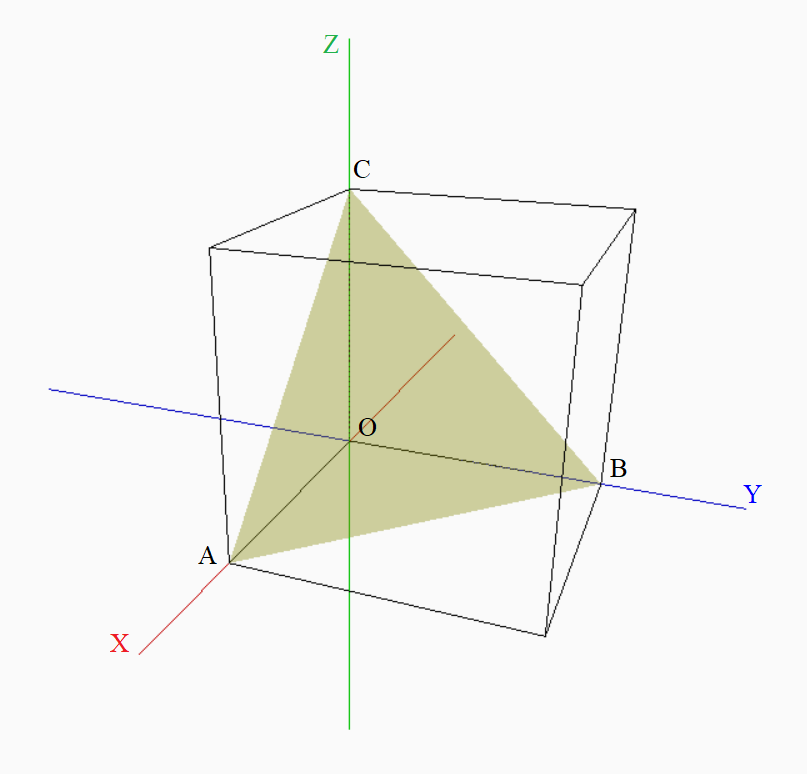


Figure 9: Millers indices using the program

After getting intercepts we can draw the plane using beginShape() function:

beginShape();

vertex(x1,y1,z1);   //vertex 1

vertex(x2,y2,z2);   //vertex 2

vertex(x3,y3,z3);   //vertex 3

endShape();

which draws the plane joining corresponding 3 point.

**2.7 Fourier Series**

Fourier series is expansion of a periodic function in terms of trigonometric series, i.e. any periodic function with period can be written as

where,

We can understand Fourier Series expansion intuitively by making use of rotating circles of different frequencies having radii equal to that of coefficients in Fourier Series expansion. For example, consider , here we have taken first two terms of the expansion. Using rotating circles, we can represent this as shown in the fig.

Here the first circle is of radius and the with rotating radius vector, the second circle has radius of length and the radius vector of this circle starts where the radius vector of first circle ends. If radius vector of first circle rotates with frequency then radius vector of second circle rotates with frequency .

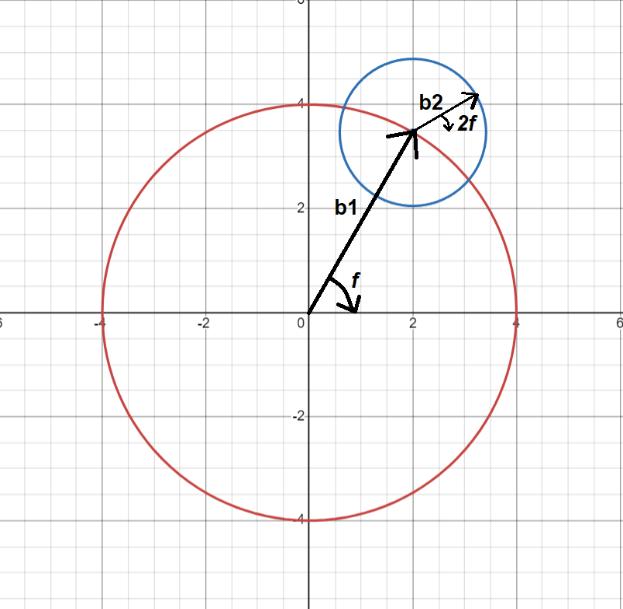


Figure 10: Radius of red circle equals first term in series  
expansion and radius of second term equal to second  
term in series expansion with vector b2 rotating with twice frequency that of b1.

The program currently consists of two functions

**1. Square Wave**

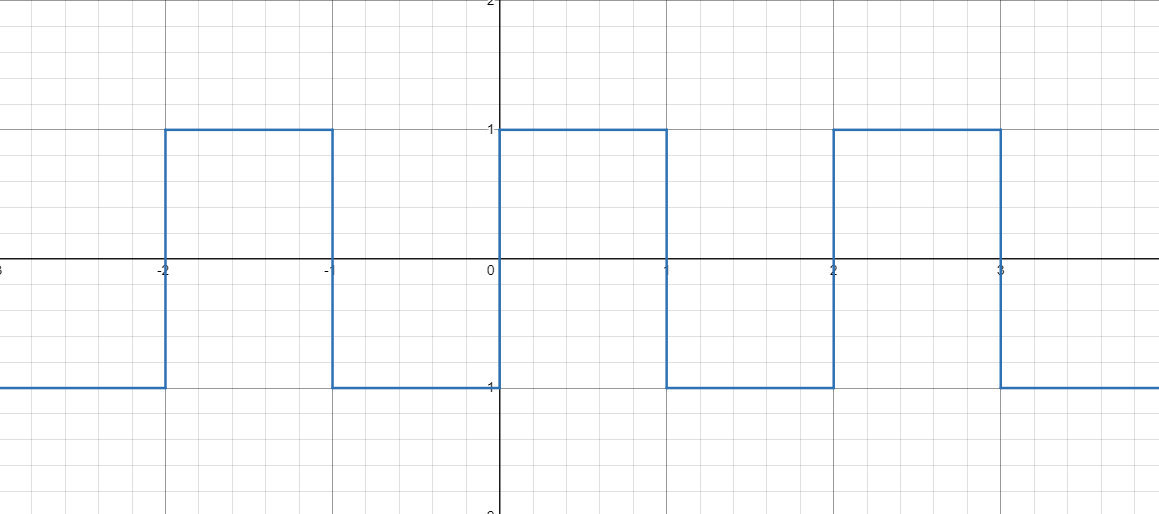
****

Figure 11: Square Wave

In the program I have used giving the Fourier Coefficients as

**2. Saw Wave**

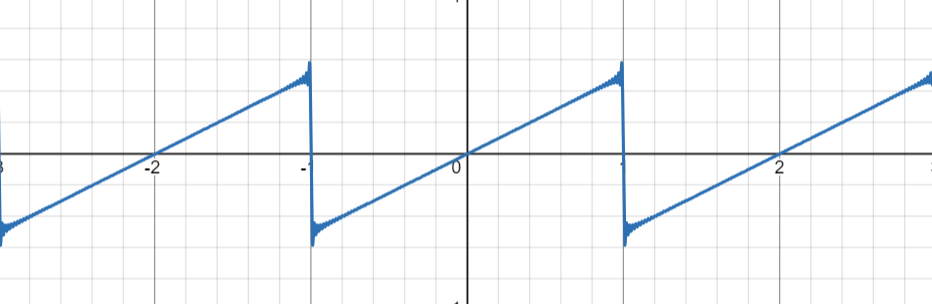


Figure 12: Saw Wave

In this case and give the corresponding Fourier Coefficients as

**2.8 Vector Field Plotter**

Perhaps the most distinctive model in the project and most intricate one, aims to visualize the vector fields with colors representing the strength of the field at that point which is a remarkable feature missing in other websites so far.

Consider a vector field

The magnitude of the field is given by

The Zenith angle and Azimuthal angle (refer to fig.6) is given by

The program takes vector field components as input from the user along with other parameters such as scale, arrow size, density and dimensions and then assigns a vector (an arrow) to an array of locations withing the space (2D or 3D).

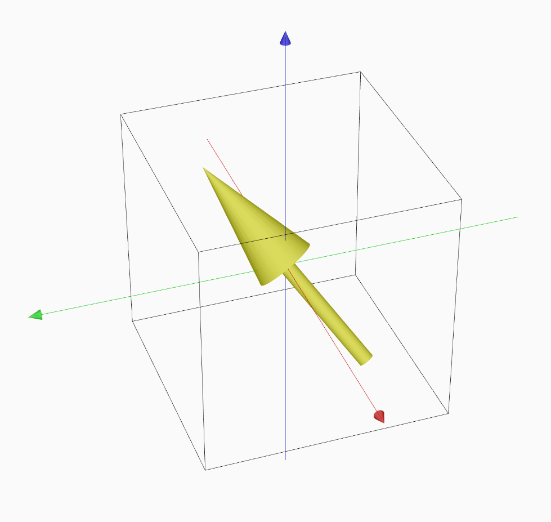
Most crucial part of this model is to make an Arrow class used as a tempelate to create arrow objects. It creates the arrows which represents the direction of vector field at a point. To each vector its variables like position (x, y, z), Magnitude, Zenith angle, Azimuthal angle, components and size are assigned. The arrow is rotated using rotateX() and rotateZ() functions passing them values of Zenith and Azimuthal angles respectively.

Figure 13: Single arrow representing direction of  
vector field at that point

The equation of field is taken from the user as string inputs in the control panel and is parsed as JavaScript math object using the Parser. The parser for the same is taken from the Git Hub repository “js-expression-eval”. Here is an example of working of the parser:

expr = Parser.parse("2 \* x + 1");

((2\*x)+1)

expr.substitute("x", "4 \* x");

((2\*(4\*x))+1)

expr2.evaluate({ x: 3});

25

Using constructor of Arrow object, arrows are set at an array of the points. All of the arrows are stored in an array in the program.

Figure 14: Array of vectors made from arrows representing vector field

After setting up of the vector field Magnitude of all vectors in stored in Magnitude array. Vectors with infinite magnitude are ignored and are also not displayed in the program. After this process the Color is assigned to each vector. To map magnitude to color, I have used the following technique.

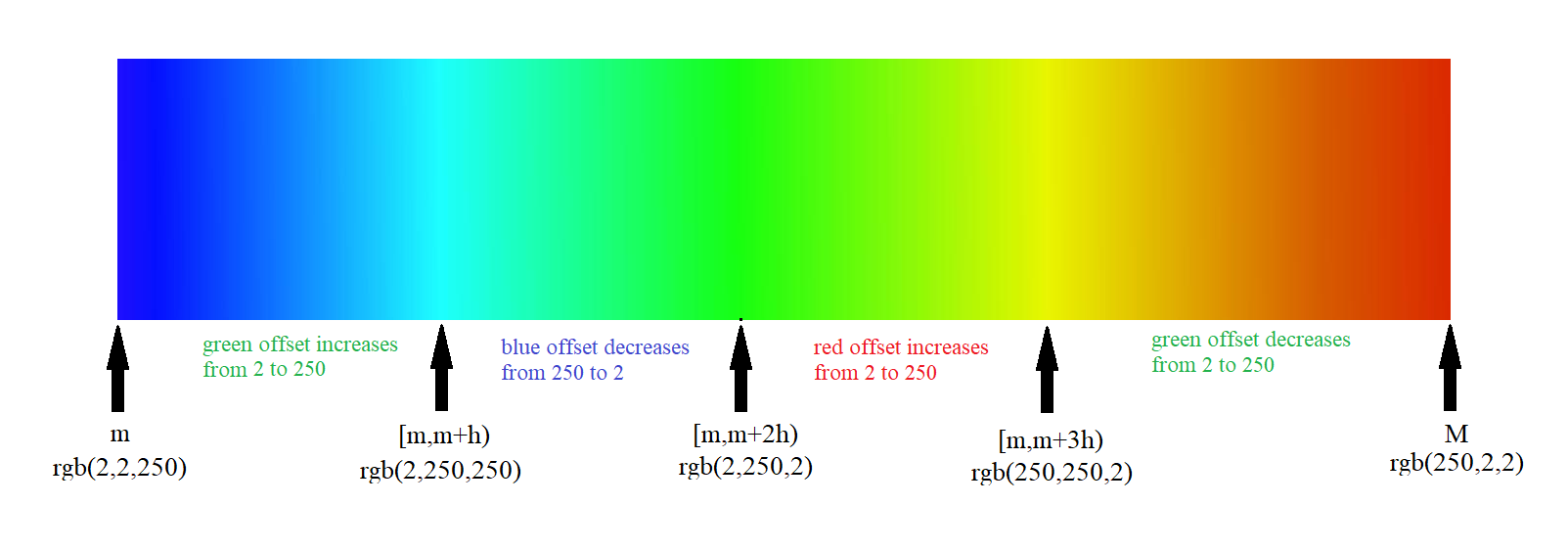
Firstly, the maximum and minimum values of the Magnitude array are taken. Let and be the maximum and minimum values respectively. Then the whole interval is divided in four parts; , , and .

Figure 15: Representation of working of color mapping

The rgb() function helps to assign the any color in by providing the red, green and blue values as argument. The mapping is done as shown in the figure above using the map() function in p5js.

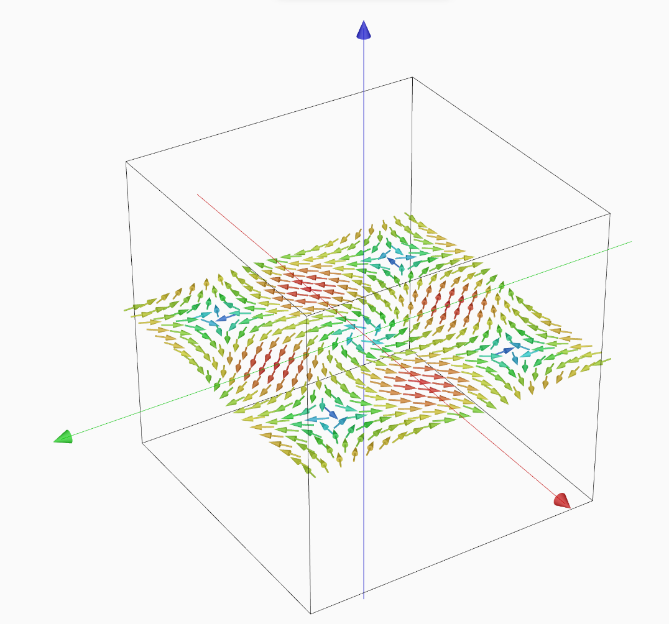
color = map(value, start1, stop1, start2, stop2)

Figure 16: Vector field after color mapping

After assigning the color it is stored in color array, then vectors are drawn using the display() function made in Arrow class.

**2.9 Lissajous Figures**

The patterns, that are traced by a particle which is subjected simultaneously to two perpendicular Simple Harmonic Motions (SHMs) of different frequencies, are known as Lissajous Figures [10].

For example, if we superpose two perpendicular SHM’s of frequencies in ratio 1:2. The two SHMs are given by

Then its resultant is given by:

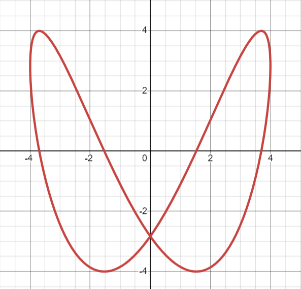
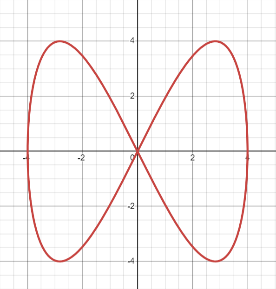
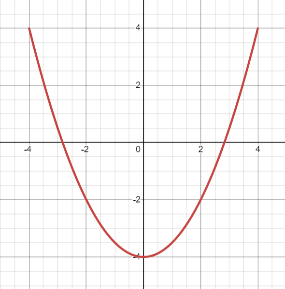
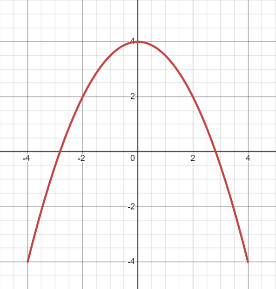
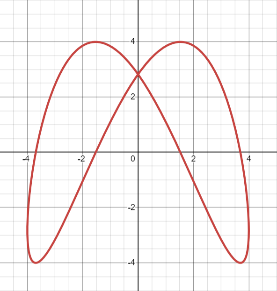
For different values of we can have different figures as shown below:

Figure 17(a):

Figure 17(a):

Figure 17(a):

Figure 17(a):

Figure 17(a):

similarly, we can have a variety of figures for different frequency ratios.

We can also analyze the motion using graphical method (see fig. below). Circle with center has rotating vector rotating with frequency and circle with center has rotating vector rotating with frequency and additional phase . The resultant is drawn for different values of time giving the resultant Lissajous figure.

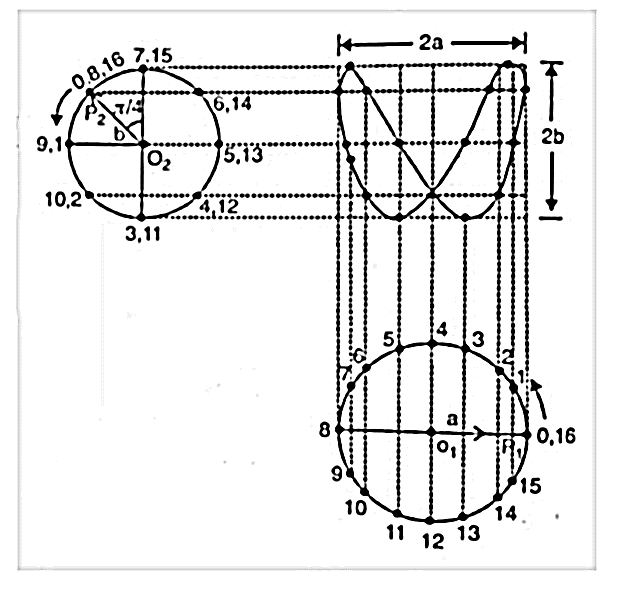


Figure 18: Graphical Method to generate Lissajous Figures

The most general form of two perpendicular SHMs can be given by

And the program uses the same above graphical method to draw Lissajous figures, where user can enter the desired frequency and and phases and to get the corresponding figure. Meanwhile, the constats and are kept same and constant and user cannot change them, moreover, they don’t make much difference in understanding the concept.

**3. Program**

**3.1 Algorithms and Flowcharts**

**3.1.1 Solid State Physics Models**

Start

- Declare Variables

- Set up local storage for orientation and state restoration

- Define canvas size and properties

- Define functions for drawing shapes and handling events

- draw()

- Set up canvas background

- Get user input for lattice type

      - Draw lattice based on user input

      - orbit()

            - Handle mouse orbit movement

            - Handle zoom using mouse wheel

      - makeSphere(x, y, z, radius)

            - Draw a sphere at specified position with given radius

- Various functions for drawing different lattice types (e.g. SimpleCubic, BodyCentered, etc.)

-  MillerIndices()

    - Draw plane corresponding to h, k, l values provided by user

      - makeBox(len)

            - Draw a box with given length

      - CsCl()

            - Draw CsCl lattice

       - NaCl()

            - Draw NaCl lattice

- Other functions for drawing different lattice voids (e.g. triangularVoid, tetragonalVoid, etc.)

End

Here is the file arrangement of the Program

- CSS

- style.css

- JavaScript

- processing.js

- main.js

- index.html

The index.html is the main html file providing structural background of the project the user input fields are defined here. The file contains the <canvas> element into which our main code of p5.js is sketched.

The style.css file contains the code for design of the web-page.

The Processing.js file contains the code to enable the usage of p5.js functions in the JavaScript. It is the main file allowing us to use p5.js in the Java

Main code of models is contained in main.js. The program first sets up the required variable. Then some of the utility and orientation related variable are stored in the local storage of the browser to restore the orientations and state of the program after the reload or opening the program in Fullscreen.

The orbit function plays a crucial role in the program setting up the camera for viewing the scene. As discussed in sec. 2.5.1 it allows to rotate the scene using mouse movment and zoom into it using mouse scrolling, hence enabling user to interact with the program.

The program contains many other functions to draw the corresponding crystal structure and the miller indices. The draw function loops itself forever till the program is closed by the user each time drawing the output corresponding to the changes made.

**3.1.2 Fourier Series**

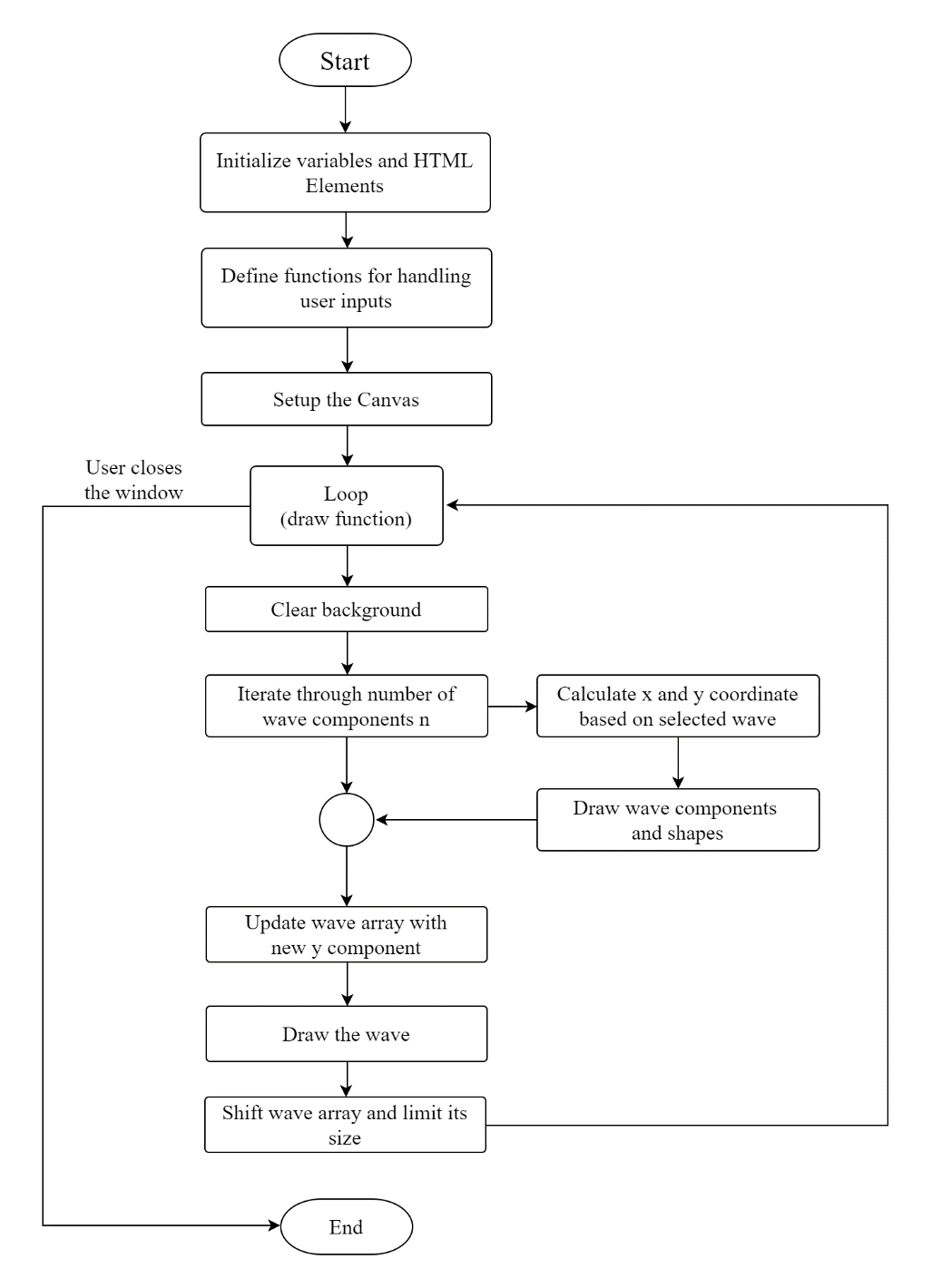
****

Figure 19: Flowchart of Fourier Series Model Code

Here is the file arrangement of the Program

- Images

- contains images required in program

- Libraries

- p5.min.js

- p5.sound.min.js

- index.html

- sketch.js

- style.css

The program initializes variables such as time (representing the current time), **wave** (an array to store the y-values of the plotted wave), and slider (a slider element in the HTML document for adjusting the number of terms in the Fourier series).

The user can select between two types of input functions (e.g., square wave or sawtooth wave) using dropdown menus in the HTML document. After which setup() functions sets the canvas for drawing.

The draw() function is the main loop of the program it draws the Fourier series visualization on the canvas. The slider value decides the number of terms in Fourier series expansion after which the x and y coordinate of the wave is calculated using a for loop iterated n number of times.

The corresponding x and y values are added in the wave array which are used to draw the wave using beginShape() and endShape() functions. The time variable is incremented to animate the wave. After this the length of wave array is checked, if it exceeds the preset threshold value the old values are removed to keep the visualization smooth.

**3.1.3 Vector Field Plotter**

Start

- Declare Variables in global.js

- Set up local storage for orientation and state restoration

- Create Control Panel in HTML file index.html

- Define canvas size and properties

- draw()

- Set up canvas background

- check if dimension is 2D or 3D

- create 2D or 3D vector, magnitude, color array based on dimension (handled

by control.js)

      - make arrow object through Arrow() constructor and store it in array

- Arrow()

- setup coordinate (x, y, z), arrow size, vector field component,  
 magnitude, azimuthal and zenith angle.

      - if Control Panel is not click then only allow navigation function

      - orbit() (function in orbit.js)

            - Handle mouse orbit movement

            - Handle zoom using mouse wheel

- else check for change in scale, arrow size, density and vector components  
 if there is change perform corresponding operation

- draw Axes and box

      - based on dimension run display() function on all vector array element

            - draw the arrow at specified coordinate

- if window is reloaded run resizeHandler() handled by resizeHandler.js

End

Here is the file arrangement of the Program

- Images

- contains images required in program

- Libraries

- p5.min.js

- p5.sound.min.js

- controls.js

- drawing.js

- global.js

- orbit.js

- parser.js

- resizeHandler.js

- index.html

- sketch.js

- style.css

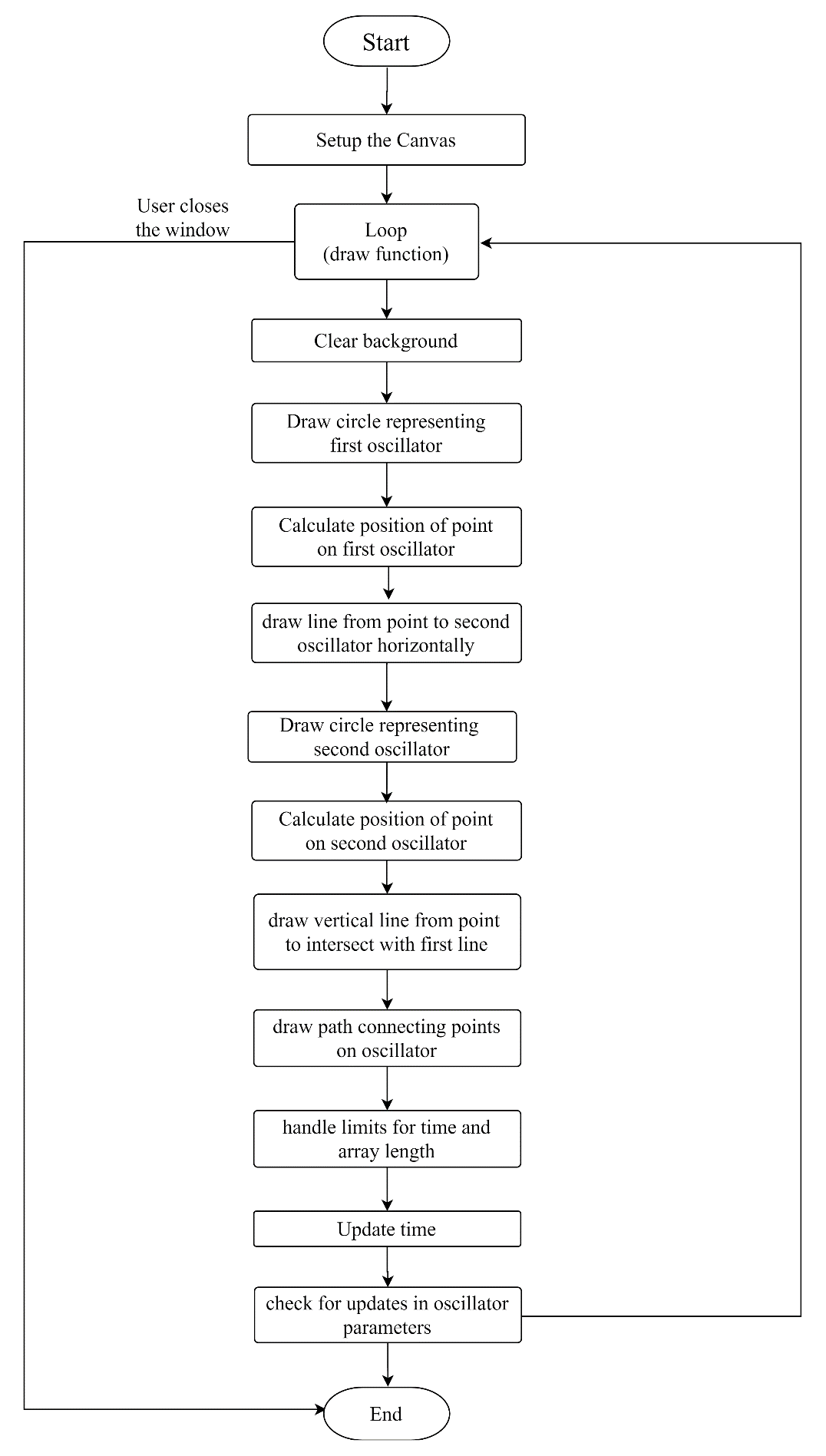
**3.1.4 Lissajous Figures**

Figure 20: Flowchart of Lissajous Figure Model Code

Here is the file arrangement of the Program

- Images

- contains images required in program

- Librarires

- p5.min.js

- p5.sound.min.js

- index.html

- sketch.js

- style.css

The program initializes variables such as canvas dimensions (width and height), circle radii (r1 and r2), frequencies (w1 and w2), phase differences (d1 and d2), and time variables (t1 and t2). It also initializes arrays x and y to store coordinates for plotting.

The setup function creates the canvas of specified width and height.

The draw() function is the main loop of the program. It draws the Lissajous figure on the canvas.

The canvas background is set to black

The first circle is drawn with radius r1. A small circle is drawn on its circumference based on the current time t1. The position of this circle is calculated using trigonometric functions.  
Similarly a second circle is drawn with radius r2 with a small circle drawn on circumference based on time t2 with its position also determined using trigonometric functions.

The Lissajous figure is drawn using beginShape() and endShape() functions, connecting points stored in the arrays x and y. The time variables t1 and t2 are decremented by rate to animate the Lissajous figure.

The refetch() function retrieves values from HTML input elements and updates variables accordingly. This allows users to change the frequencies, phase differences, and animation rate of the Lissajous figure interactively.

**4. Results and Discussion**

**4.1 Structures and Test Cases**

**4.1.1 Solid State Models**

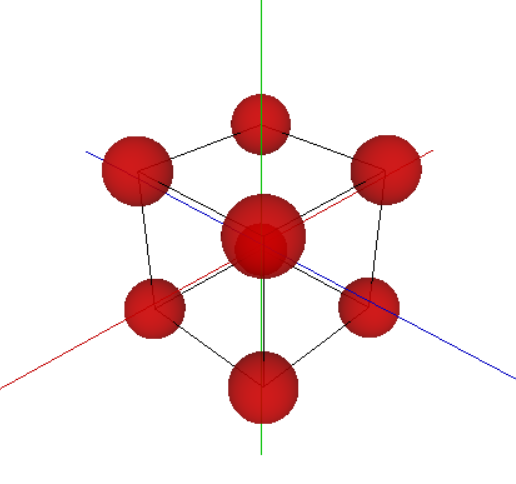
****The program sketches interactive array of models of unit cells, crystal structure, void and Miller’s indices satisfactorily. Moreover, the orbit function allow to zoom and view the structure from any orientation making it easier for both educator and learner to grasp the topic quickly and easily.

Figure21 (b): Simple Cubic Cell

Figure 21(a): Simple Cubic Cell

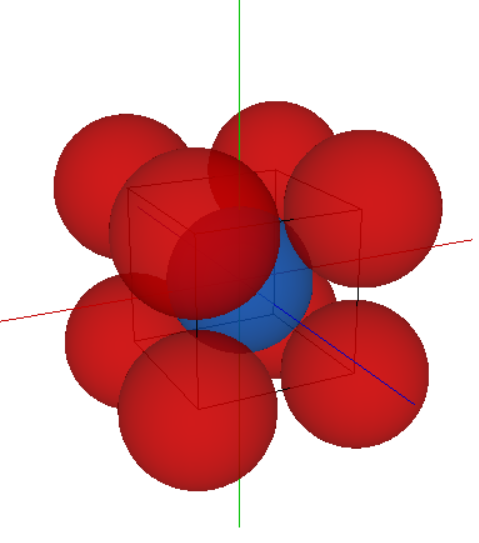
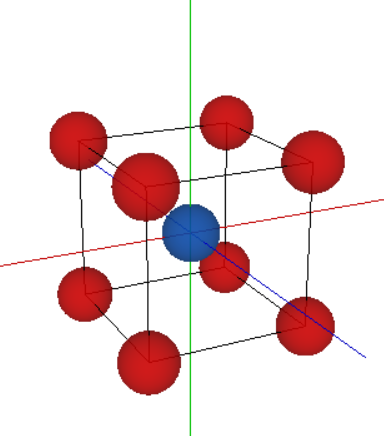
****

Figure 22(b): Body Centered Cell

Figure 22(a): Body Centered Cell

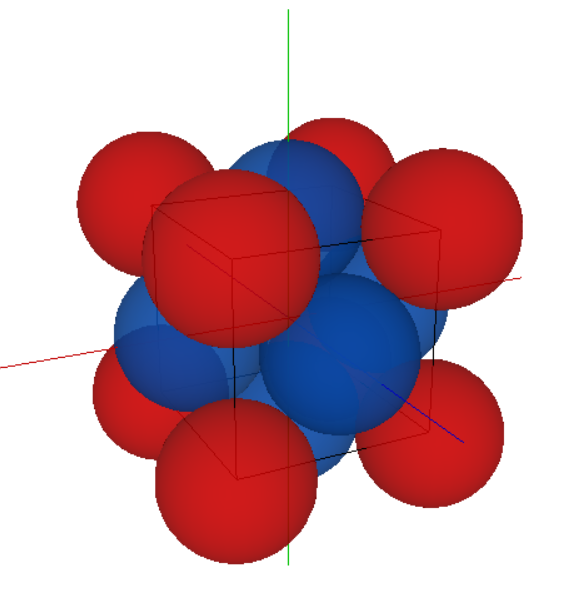
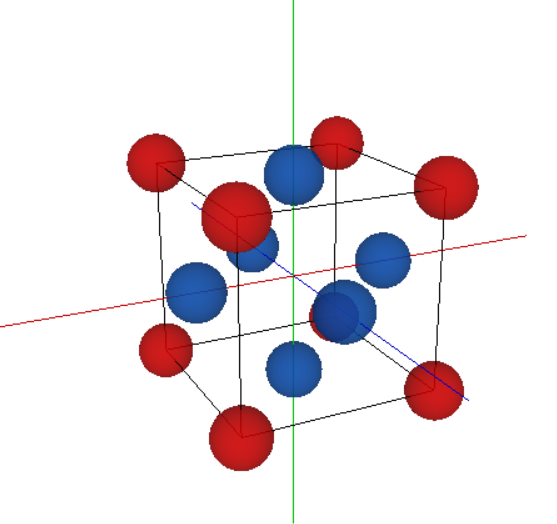
****

Figure 24(a): Face Centred Cell

Figure 23(b): Face Centerd Cell

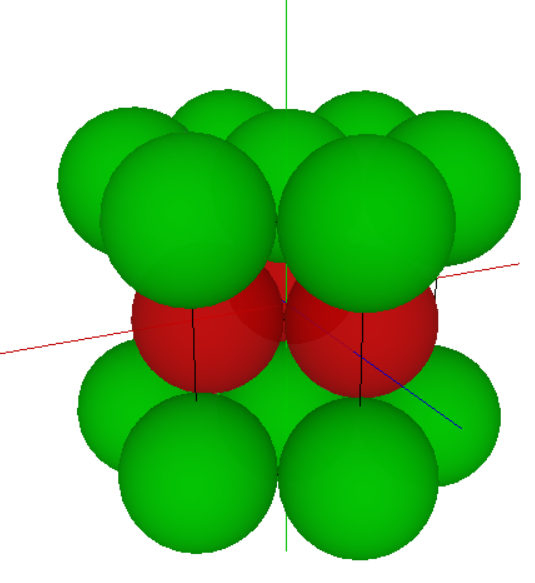
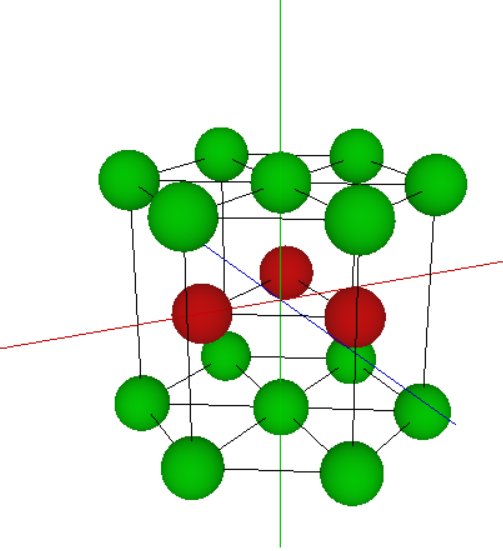
****

Figure 26(b): Hexagonal Unit Cell

Figure 25(a): Hexagonal Unit Cell

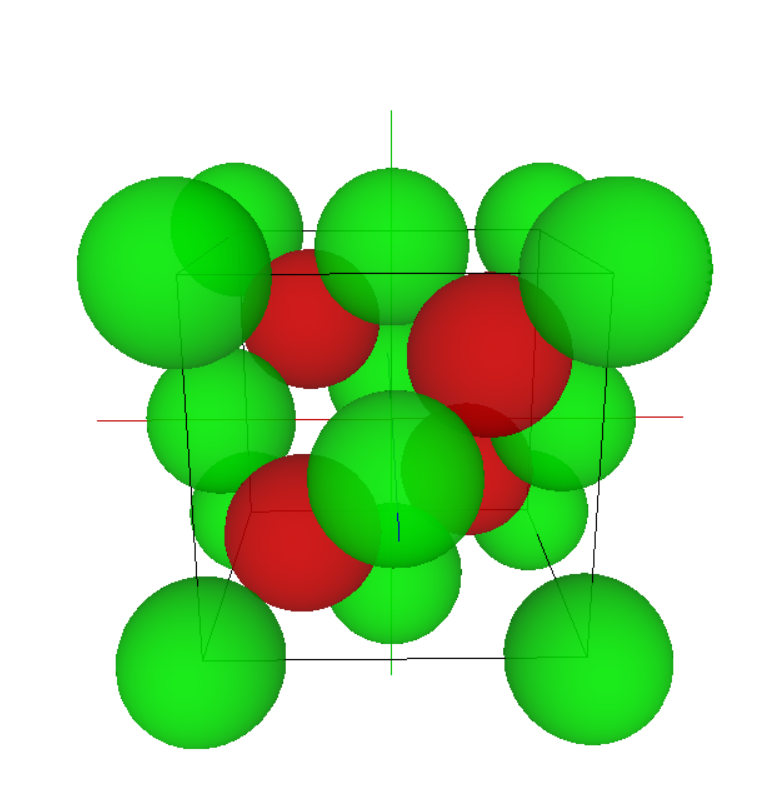
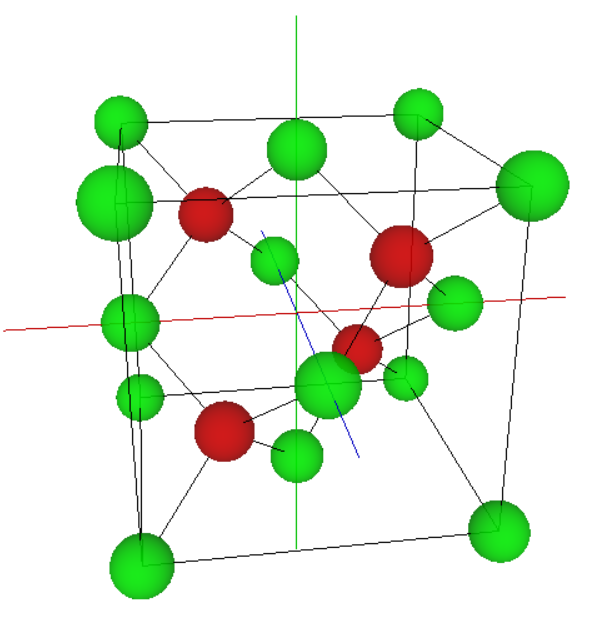
****

Figure 27(a): Diamond Cubic Cell

Figure 27(b): Diamond Cubic Cell

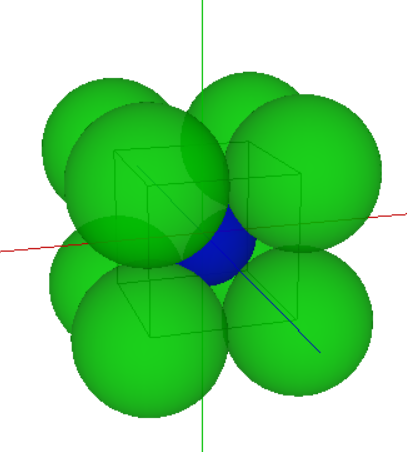
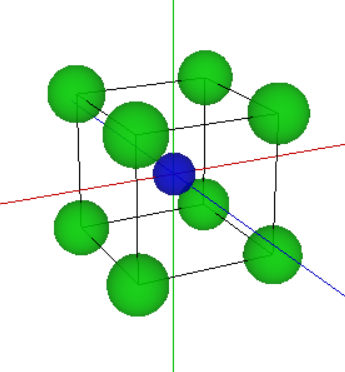
****

Figure 28(b): CsCl

Figure 28(a): CsCl

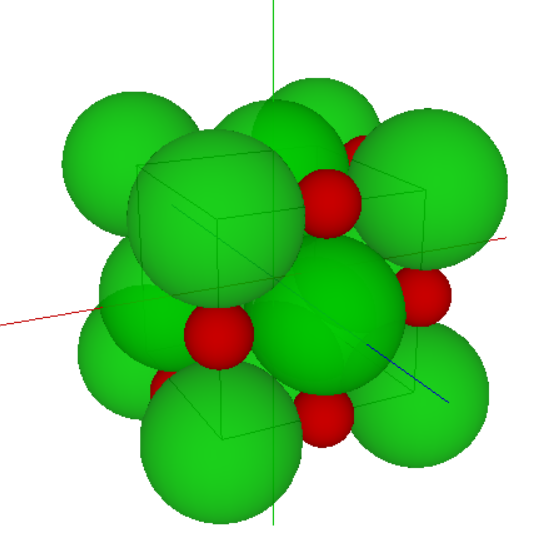
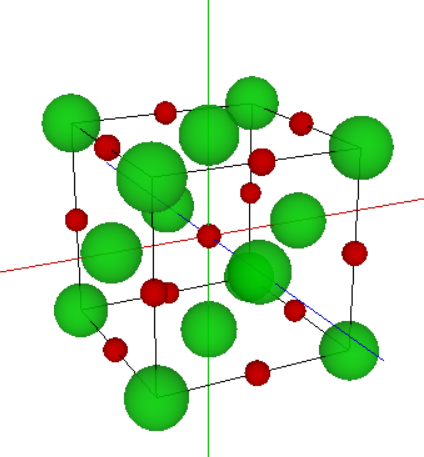
****

Figure 29(a): NaCl

Figure 29(b): NaCl

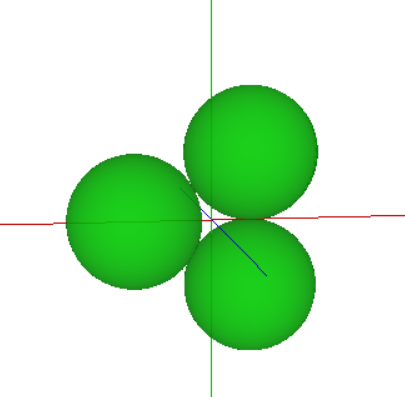
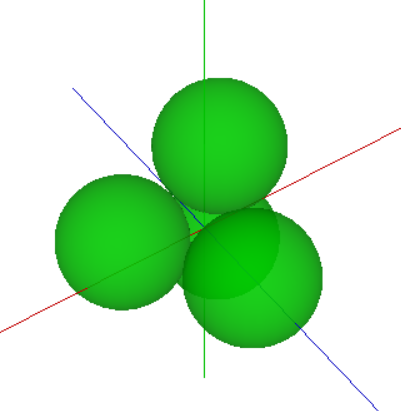
****

Figure 30(b): Tetrahedral Void

Figure 30(a): Triangular Void

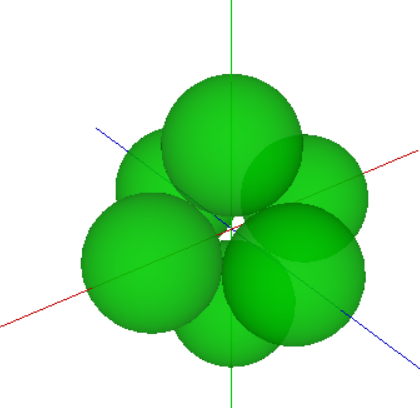
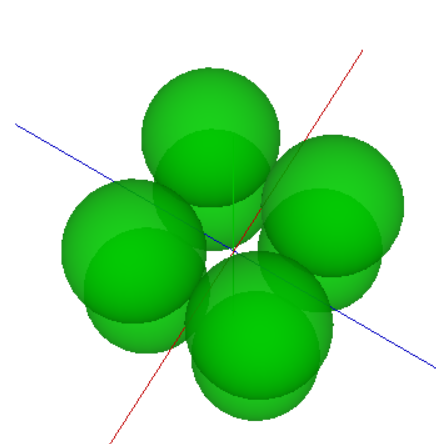
****

Figure 31(a): Cubic Void

Figure 31(b): Octahedral Void

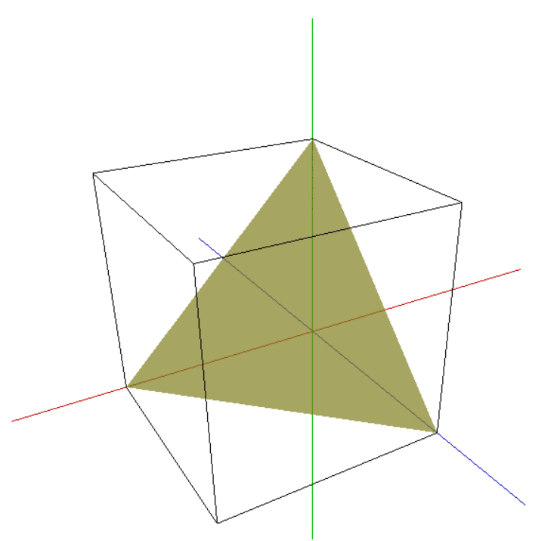
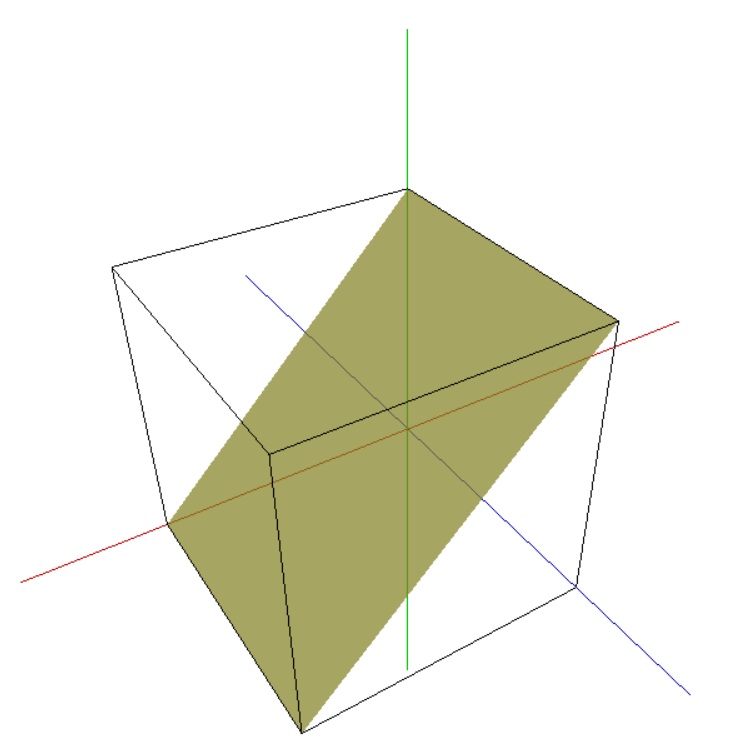
**  
Test Cases:**X axis in red, Y axis in blue and Z axis in Green.

Figure 32(a): Plane corresponding to   
Miller indices (1 1 1)

Figure 32(b): Plane Corresponding to   
Miller indices (1 0 1)

**4.1.2 Fourier Series**

The Fourier Series program serves as a powerful tool for intuitively grasping the concepts of harmonic frequencies and coefficients in Fourier series expansions. It provides users with an interactive platform to select the number of terms in the series, offering remarkable insights into how the series progressively approximates the expansion function as the number of terms increases. Through visualizations and dynamic adjustments, users gain a deeper understanding of the principles underlying Fourier series, making complex mathematical concepts accessible and comprehensible.

Let us consider the two test cases for each square and saw wave. The output of Desmos Graphing Calculator [11] is used as reference.

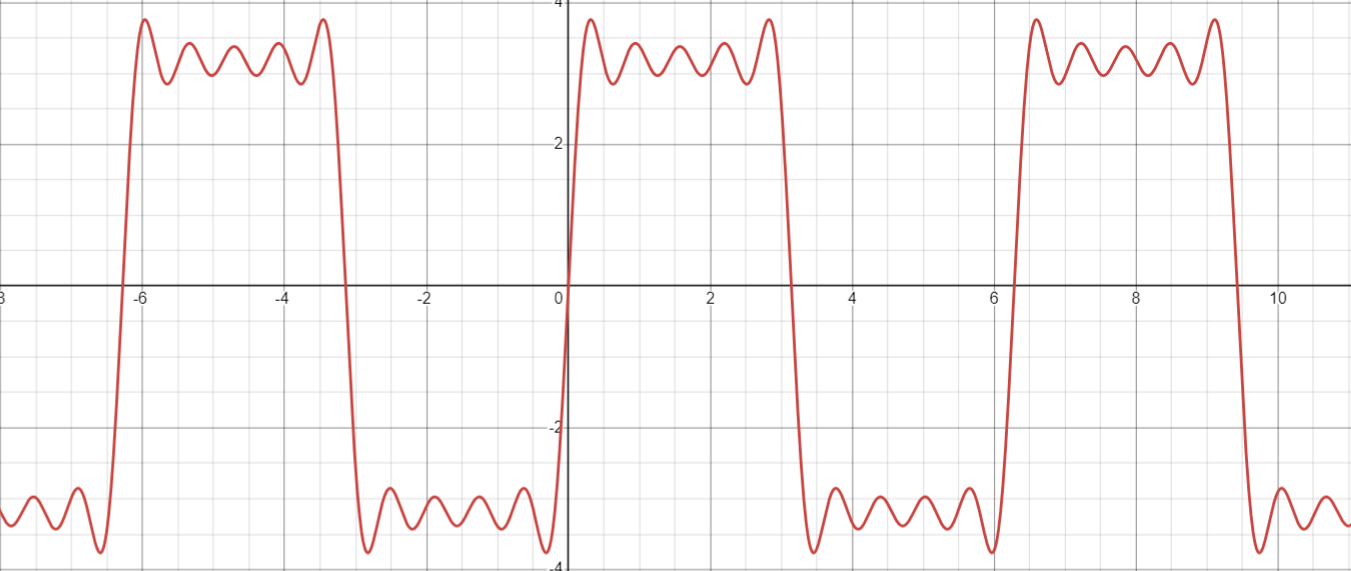
**Test Case 01:**

Figure 33(a): Expected Output

terms

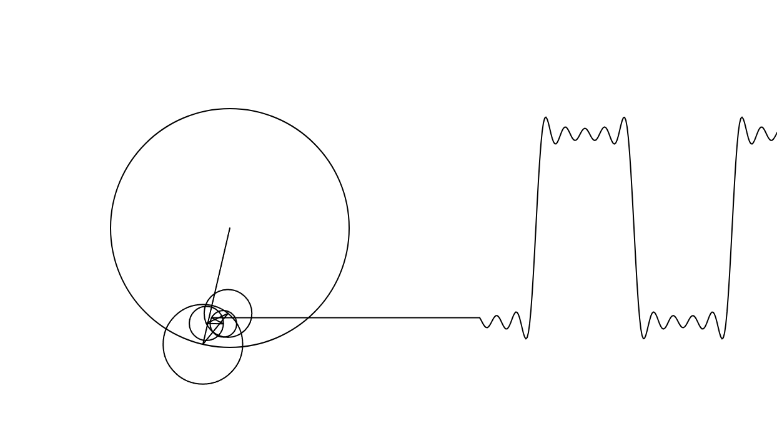
****

Figure 33(b): Given Output

**Test Case 02:**

terms

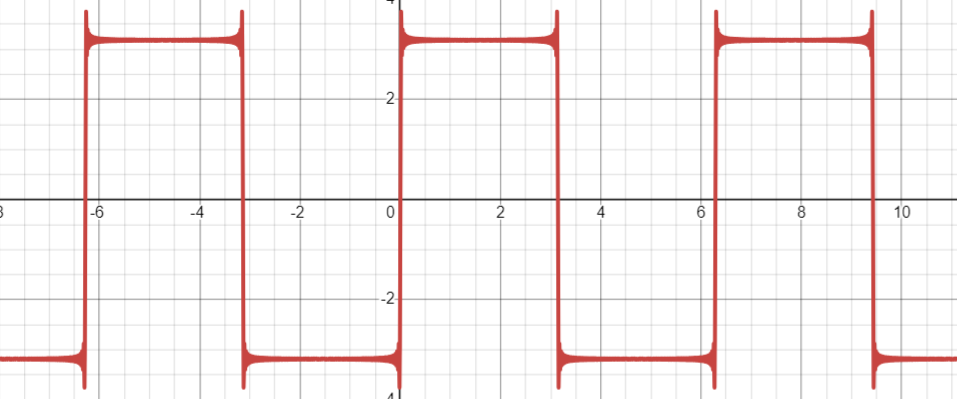
****

Figure 34(a): Expected Output

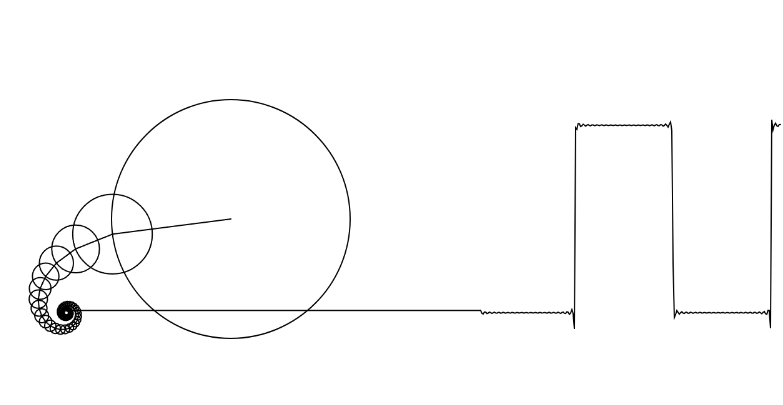
****

Figure 34(b): Given Output

**4.1.3 Vector Field Plotter**

The Vector Field Plotter will allow the user to plot any 2D or 3D vector fields, enabling to visualize the vector fields and study of their nature. The user can enter any vector field through the control panel and adjust the scale, arrow size and density of the field.

Although the program has a bug of simultaneous rotation of vector field and control panel in some unknown circumstances. Hence a RESET button is provided in control panel. Increasing the number of vectors in 3D mode also reduces the latency of the program is a downfall.

Here are some of the test cases. I have used GeoGebra vector field plotter [12] [13] as reference.

**Test Case 01:**

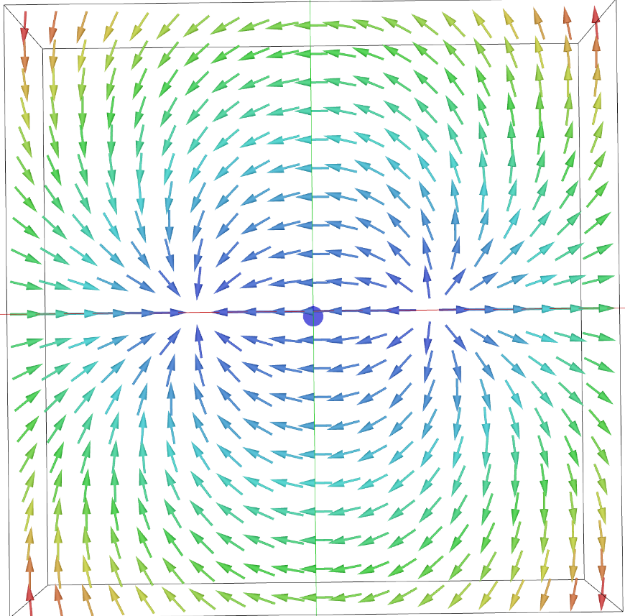
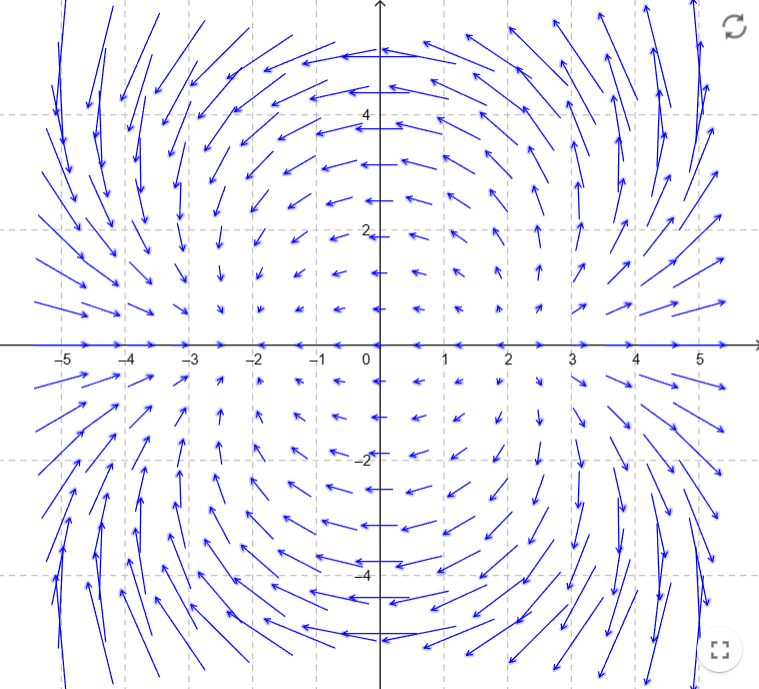
****

Figure 35(b): Given Output

Figure 35(a): Expected Output  
Image Credits: Juan Carlos Ponce Campuzano

**Test Case 02:**

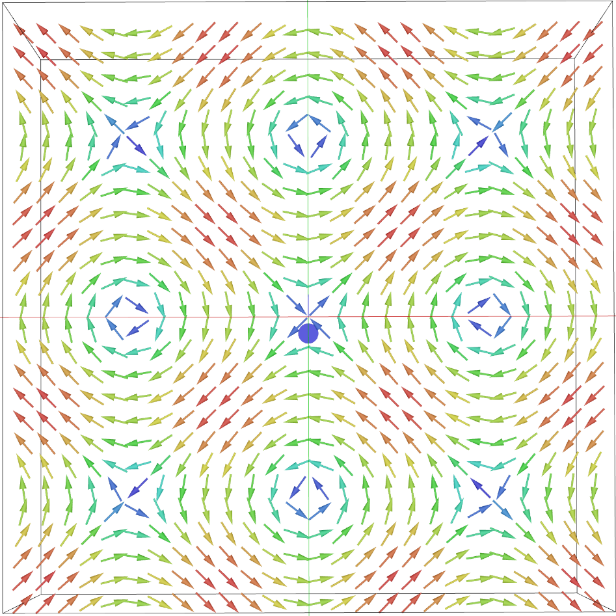
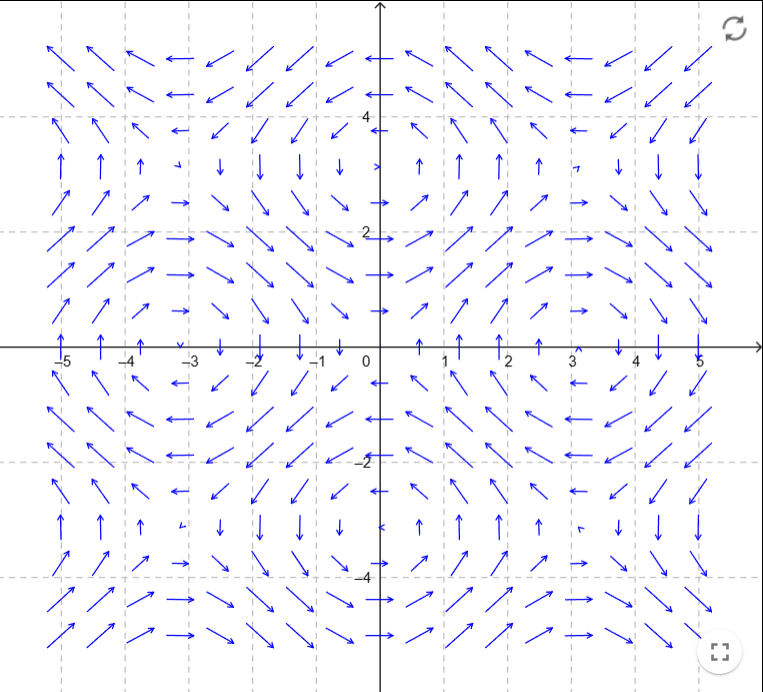
****

Figure 36(b): Given Output

Figure 36(a): Expected Output  
Image Credits: Juan Carlos Ponce Campuzano

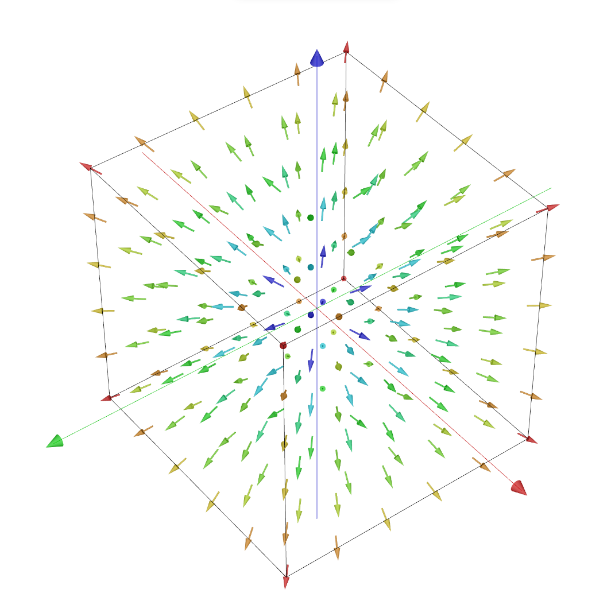
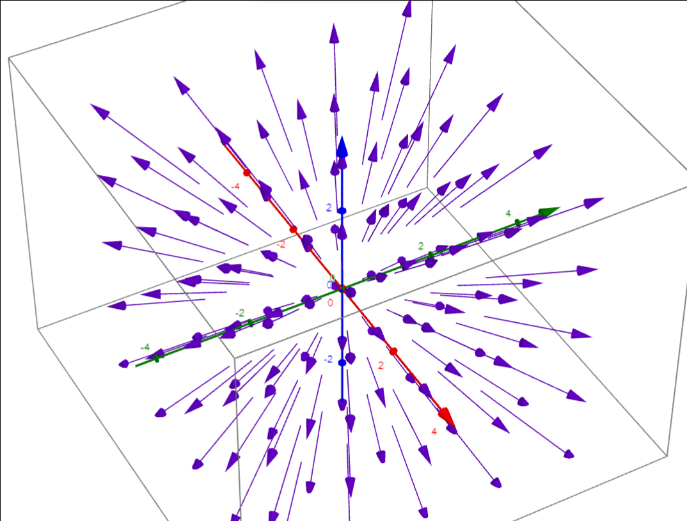
**Test Case 03:**

Figure 37(b): Given Output

Figure 37(a): Expected Output  
Image Credits: Juan Carlos Ponce Campuzano



**4.1.4 Lissajous Figures**

Plotting Lissajous figures using a geometrical approach not only facilitates a better understanding of the topic but also simplifies comprehension. This program enables users to generate Lissajous figures by inputting frequencies and phase differences, offering an intuitive visualization of these intricate patterns. By interacting with the program, users can explore various combinations of frequencies and phase differences, gaining insight into how these parameters affect the resulting Lissajous curves. Through its user-friendly interface and dynamic visualizations, the program enhances learning by providing a hands-on experience that reinforces conceptual understanding of Lissajous figures.

For test cases, I have used the book “The Physics of Waves and Oscillations” by N.K. Bajaj as reference.

**Test Case 1:**

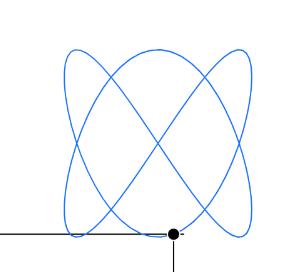
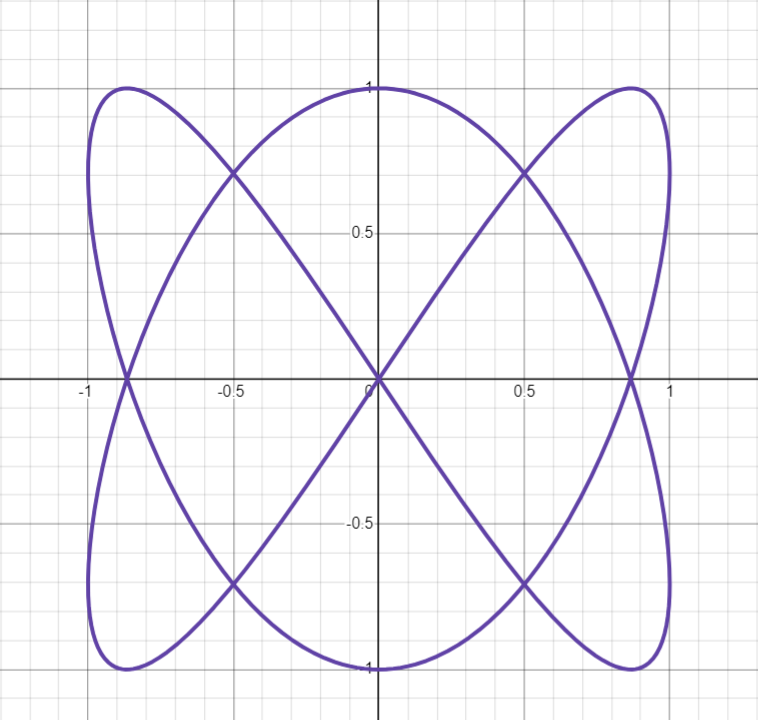
****

Figure 38(a): Expected Output

Figure38(b): Given Output

**Test Case 2:**

Figure 39(a): Expected Output

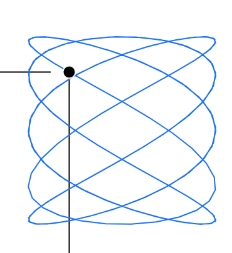
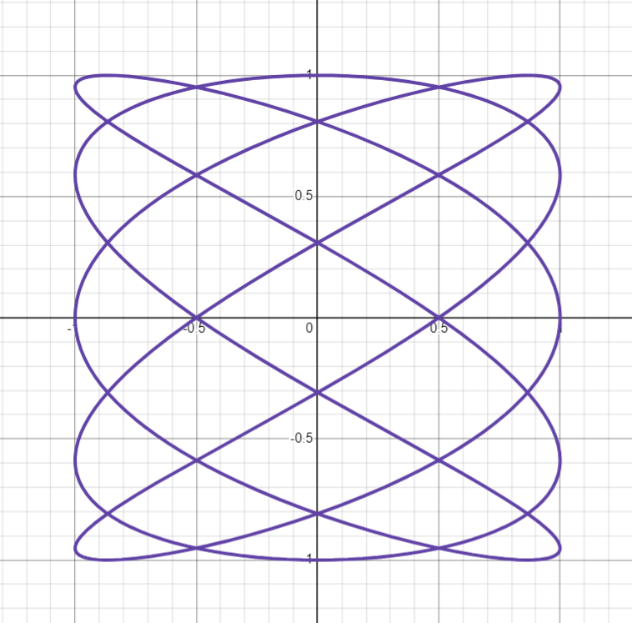
****

Figure 39(b): Given Output

**Test Case 3:**

Figure 40(a): Expected Output

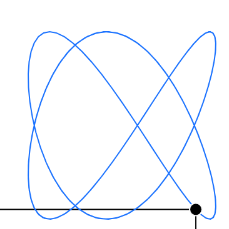
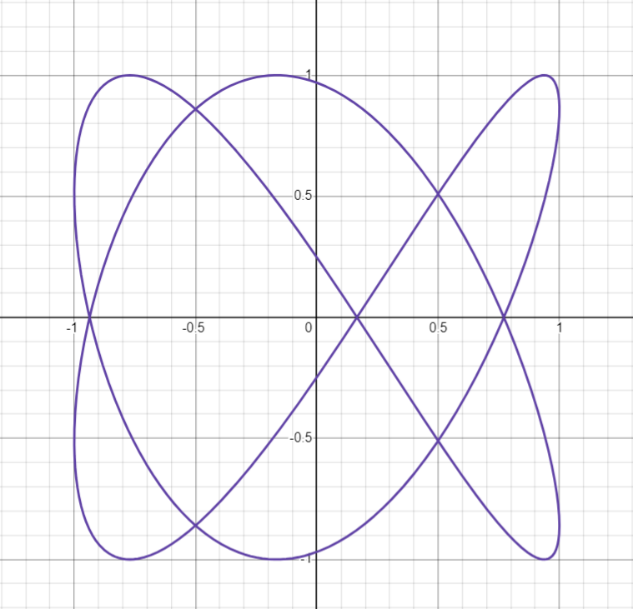
****

Figure 40(b): Given Output

**5. Conclusion and Future Scope**

**2.1 What is p5js and WEBGL**

p5.js [[2]](https://p5js.org/) is a free and open-source JavaScript library

**6. References**

1. Sanderson, G. 3blue1brown. YouTube: <https://www.youtube.com/@3blue1brown>
2. p5.js. p5.js | home. p5.js: <https://p5js.org/>
3. p5.js. Getting Started in WebGL Coords and Transform. p5.js.  
   <https://p5js.org/learn/getting-started-in-webgl-coords-and-transform.html>
4. <https://code.visualstudio.com/>
5. <https://marketplace.visualstudio.com/items?itemName=samplavigne.p5-vscode>
6. The Coding Train. (n.d.). The Coding Train. YouTube. <https://www.youtube.com/c/TheCodingTrain>
7. escopecz. js-expression-eval. GitHub. <https://github.com/escopecz/js-expression-eval>
8. p5.js. Getting Started in WebGL Appearance. p5.js. <https://p5js.org/learn/getting-started-in-webgl-appearance.html>
9. Das, H.K. (2014). *Mathematical Physics*. Seventh Revised Edition. S. Chand
10. Bajaj, N.K. (2021). *The Physics of Waves and Oscillations.* 48th Reprint. McGraw Hill.
11. <https://www.desmos.com/calculator>
12. [Vector Fields – GeoGebra](https://www.geogebra.org/m/QPE4PaDZ)
13. <https://www.geogebra.org/m/u3xregNW>