Deccan Education Society’s

**FERGUSSON COLLEGE (AUTONOMOUS), PUNE**

--|| DEPARTMENT OF PHYSICS ||--

**3D Interactive Models in Physics**

T.Y.B.Sc 2023-24

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:

**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 7. Fourier Series 8. Vector Field Plotter | **7** |
| **Chapter 3 Program** |  |
| **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.

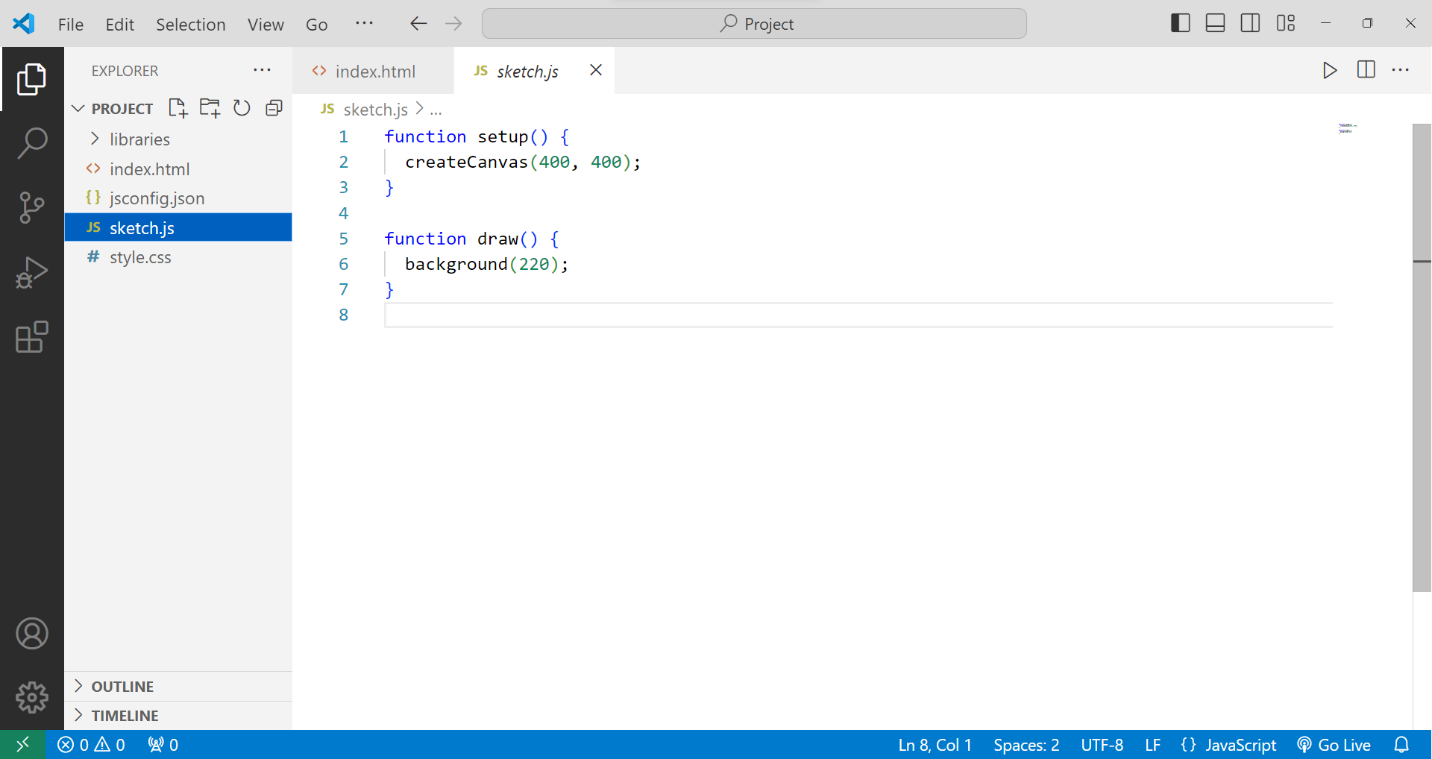


Image 1: Basic Working Environment for a p5js Model

**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

}

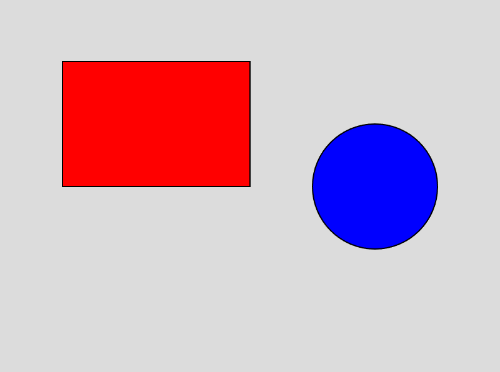


Figure 1: Output of the Previous Code

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.

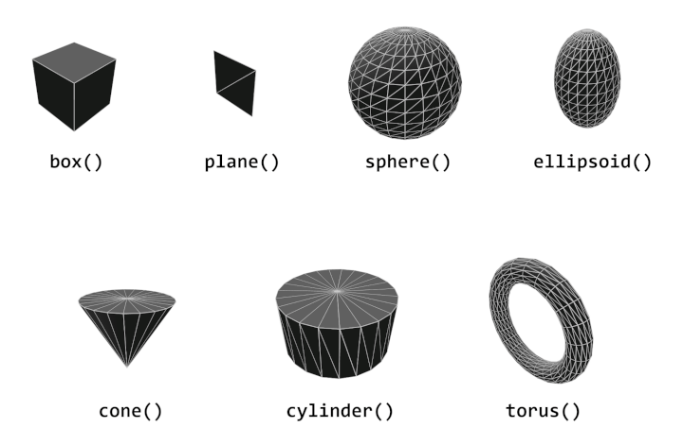


Figure 2: Objects in p5js

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);

}

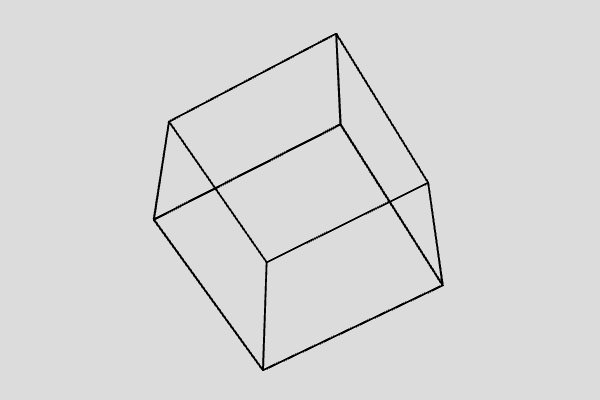


Figure 3: A 3D Cube in p5js

**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()`.

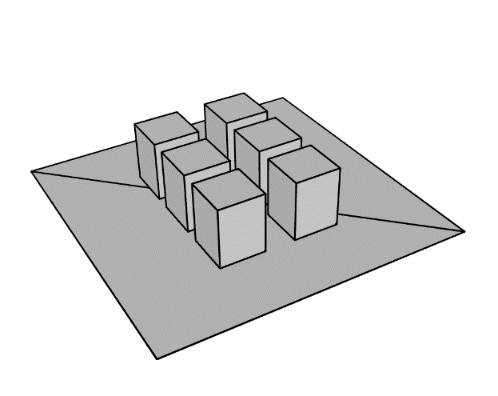
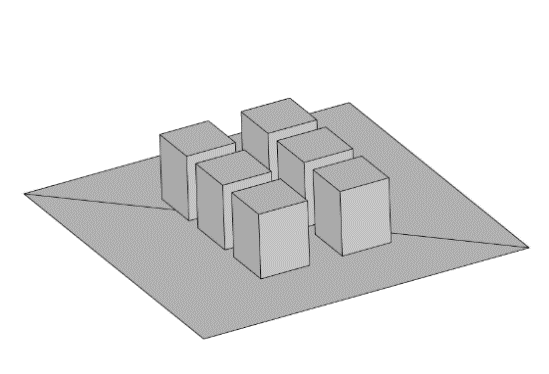


Figure 4 (b): Orthographic Projection

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

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.

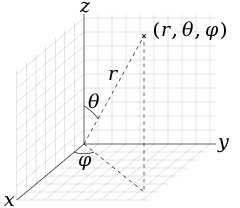


Figure 6 Spherical Polar Coordinates

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**

Lights in p5js are essential to convey the depth of the 3D structures. Hence it is necessary to use a light source in the program. A light source should be used in the draw() function. P5js allows several types of lighting in 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.
* noLights(): noLights() makes it so that all subsequent geometry is rendered without any lighting. This can be useful when you want flat, unshaded geometry.

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.

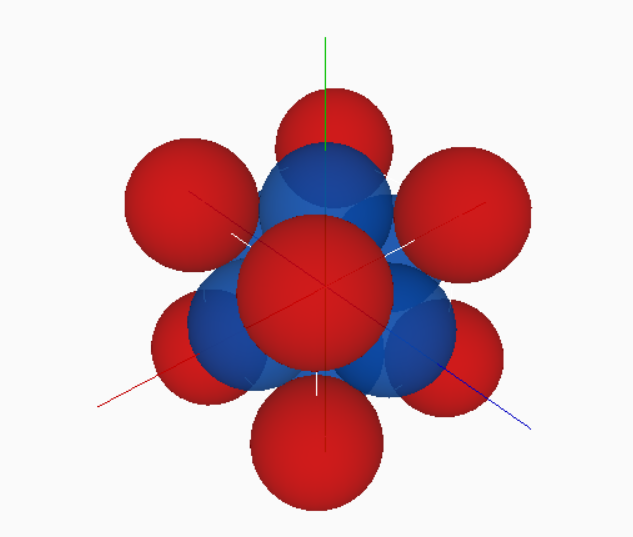
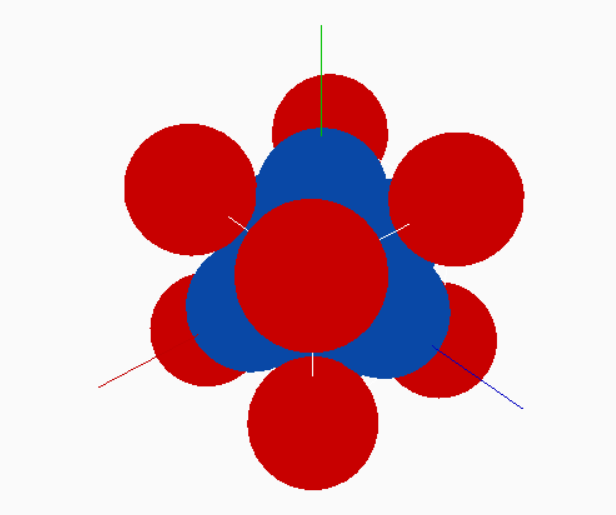
Here is an example of scene with and without lights.

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**

Objects can appear differently based on their material. Materials dictate how light interacts with the geometry and how color (or texture) gets applied to the object. Materials can be varied, making objects shiny, rough, or even textured with images.

* **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.

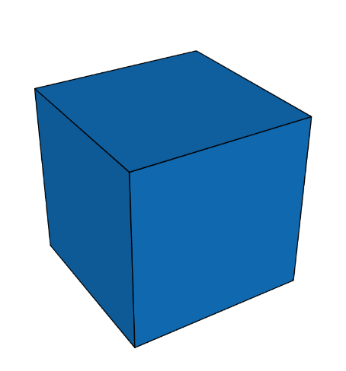
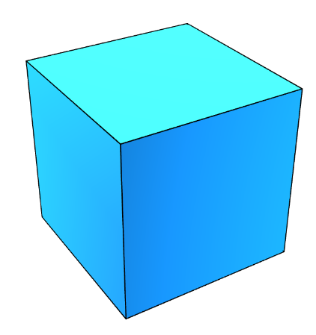
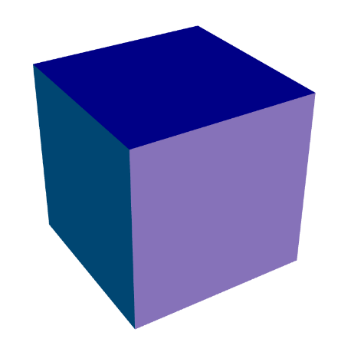
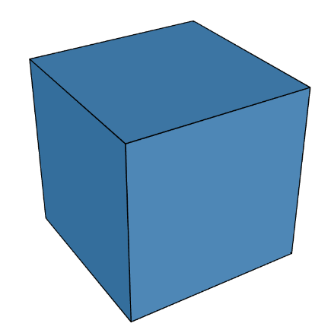


Figure 2

Figure 3

Figure 4

Figure 5

**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();

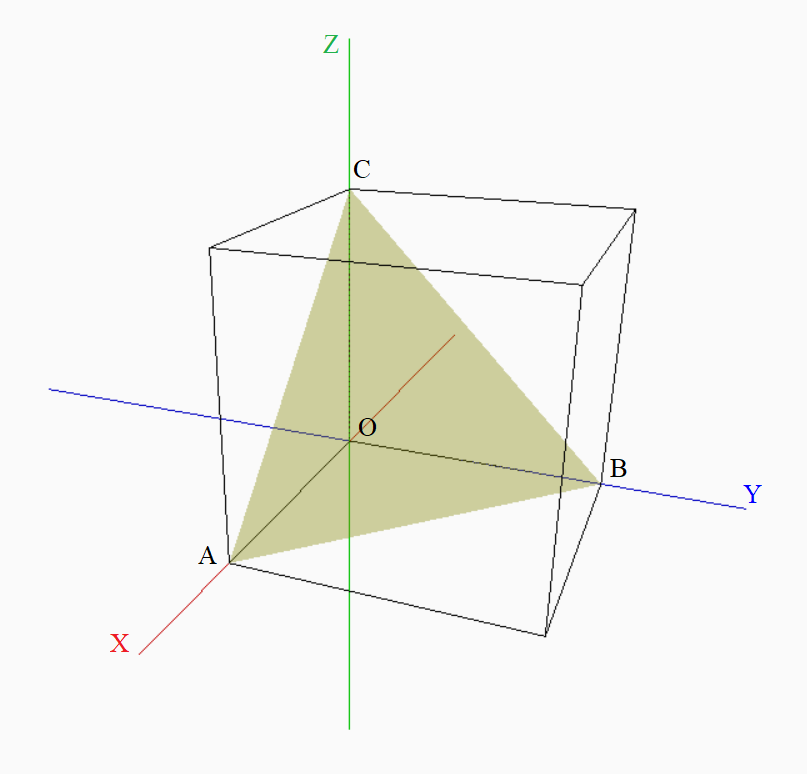
    translate(x, y, z);

    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 below:

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.

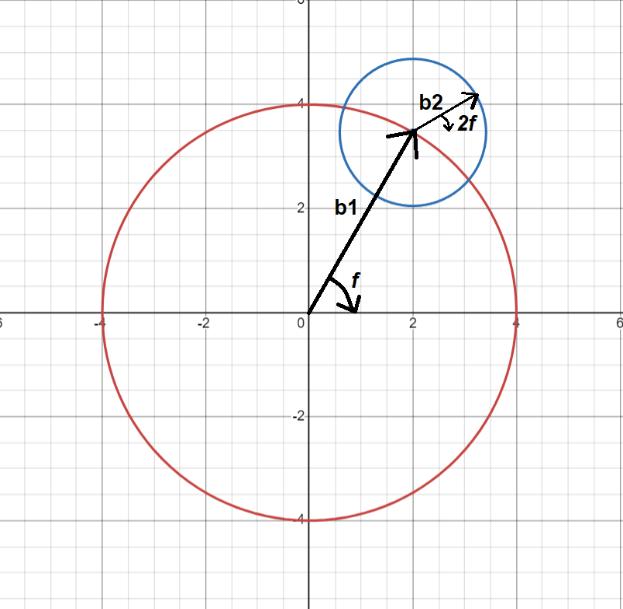
**2.7 Fourier Series**

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 .

 **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 Object which creates the arrows representing the vector field. To each vector its variables like position (x, y, z), Magnitude, Zenith angle, Azimuthal angle, components and size are assigned.

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

After setting up the Arrow object at a point using the constructer of the object, the color is assigned to the vector stored in the color array. Assigning color is done on the basis of the magnitude of the all available vectors

**3. Program**

**2.1 What is p5js and WEBGL**

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

**4. Results and Discussion**

**2.1 What is p5js and WEBGL**

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

**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.

**Brief Outline**

**1. Introduction**

**2. Theory**

* What is p5js
* Setting up working environment
* How to create objects
* Events in p5js
* What is perspective and camera
* Lights
* Material

**3. Program**

* Solid state
* Fourier series  
  1. Flow chart  
  2.
* Vector Field Plotter  
  1.

**4. Results and Conclusion**

**5. Future Scope**

**6. References**