Project Outline: Converting DSP Filters from Matlab to C and Visualizing in 3D

1. Initial Setup and Understanding of Matlab Code

• **Objective**: Understand the three Matlab-based DSP filter algorithms.

• **Actions**:

1. Review the existing Matlab code for the filters. Ensure the logic and mathematical operations of the algorithms are clearly documented.
2. Identify the types of filters used (e.g., FIR, IIR) and their functions (e.g., lowpass, high-pass, band-pass, etc.).
3. Familiarize yourself with Matlab's built-in DSP functions used in the code.

• **Deliverables**:

o A detailed description of the three filters, including their inputs, processing, and outputs.

o A clear mapping of Matlab functions to their equivalent operations in C, if applicable.

2. Design of C Code Structure

• **Objective**: Plan the C implementation of the DSP filters.

• **Actions**:

1. Choose appropriate C libraries for handling mathematical operations, such as the math.h library or DSP-specific libraries like CMSIS-DSP (if using ARM processors).
2. Plan the code structure:

▪ Create individual functions for each filter type.

▪ Define inputs (e.g., signal data, filter coefficients) and outputs (filtered signal).

3. Ensure that code is modular, allowing for easy testing of each filter independently.

• **Deliverables**:

o Flowcharts or pseudo-code outlining each filter's logic and structure in C.

o List of required C libraries.

3. Conversion of Matlab Code to C

• **Objective**: Convert each Matlab DSP filter algorithm into C code.

• **Actions**:

1. Convert the Matlab code into C line-by-line, maintaining the same functionality. Handle any matrix or vector operations using arrays and loops.
2. Implement filter-specific mathematical operations:

▪ Use convolution for FIR filters.

▪ Implement difference equations for IIR filters.

3. Optimize the C code for efficiency, minimizing memory and processing overhead where possible.

• **Deliverables**:

o Three standalone C functions (or modules) corresponding to each Matlab filter.

o A main function to load audio input data, apply filters, and store results.

4. Audio Input and Output Handling

• **Objective**: Integrate audio input/output handling in C code.

• **Actions**:

1. Implement functionality to read audio data (e.g., WAV files) in C using standard libraries like libsndfile or write a custom parser if needed.
2. Pass the audio data through the filter functions and capture the processed output.
3. Write the filtered audio data to an output file for verification.

• **Deliverables**:

o Code for reading audio files, processing them with the filters, and saving the results.

5. 3D Plotting of Filter Output

• **Objective**: Visualize the output of the filters in 3D plots (e.g., time-frequency or frequency-magnitude plots).

• **Actions**:

1. Plan the type of 3D plots to be generated, such as:

▪ Time vs. Frequency vs. Magnitude.

▪ Frequency response plots.

2. Research libraries in C or use external plotting tools that support 3D plotting, such as gnuplot, OpenGL, or integration with Python-based plotting libraries like matplotlib.
3. Write C code to prepare the filter output data for plotting:

▪ Compute the Fourier Transform (FFT) of the filtered output to get the frequency domain representation.

▪ Structure the data in a format suitable for 3D plotting.

4. Use Visual Studio to integrate the plotting functionality, leveraging libraries like OpenGL for real-time plotting or generating data to export for external visualization tools.

• **Deliverables**:

o Code for 3D plotting of the filtered signal’s frequency response.

o Example plots demonstrating the output of each filter.

6. Integration and Testing

• **Objective**: Ensure the entire workflow from filtering to visualization works

seamlessly.
• **Actions**:
1. Test each C filter function individually with various input signals and compare results against Matlab outputs for verification.
2. Test audio input and output handling, ensuring correct reading, filtering, and writing of data.
3. Test the 3D plot generation to ensure it correctly visualizes the filtered signal's properties.
• **Deliverables**:
o A fully integrated C program that filters audio data and visualizes the output in 3D.
o Test results comparing Matlab and C output.

7. Documentation

• **Objective**: Provide comprehensive documentation of the entire project.
• **Actions**:
1. Document each step of the conversion process, including any optimizations or changes made during implementation.
2. Provide user instructions for running the C code and generating 3D plots in Visual Studio.
3. Write up performance benchmarks comparing Matlab and C implementations, including execution time and memory usage.
• **Deliverables**:
o A detailed project report with code documentation.
o A user manual for running the final C program.

8. Final Review and Optimization

• **Objective**: Optimize the final C implementation and resolve any remaining issues.
• **Actions**:
1. Review the code for potential improvements, such as reducing computation time, optimizing memory usage, and improving plot rendering performance.
2. Conduct final testing on different datasets to ensure robustness and reliability.
3. Prepare the final version of the C code and ensure it is ready for delivery.
• **Deliverables**:
o Optimized C code with final 3D visualization output.
o A final report highlighting key aspects of the project and any potential future improvements.

In this project I will give you 3 codes of Matlab, and you will convert into C code with 3D plot and give me final output