3D Graph Explorer - Team Project Task Division

Project Structure Overview

The 3D Graph Explorer application can be divided into these main components:

1. **App Setup & UI Framework** - Basic Streamlit configuration, page layout, and styling
2. **Mathematical Functions & Surface Generation** - Core mathematical functions for generating 3D surfaces
3. **Visualization & Rendering System** - Plotly integration and graph display logic
4. **User Controls & Parameter Management** - Sidebar controls and interactive elements
5. **Documentation & Help System** - Information panels, usage guides, and mathematical explanations

Team Member Responsibilities

Member 1: Application Framework & UI Design

Files to Create:

- `app_framework.py` - Core application structure and main UI elements

Responsibilities:

- Set up the Streamlit page configuration
- Design and implement the overall page layout
- Create custom CSS styling for better appearance
- Implement the main header and application containers
- Design the footer and branding elements

Code Sections:

```python
# Configure page
st.set_page_config(
    page_title="3D Graph Explorer",
    layout="wide",
    initial_sidebar_state="expanded",
)

# Custom CSS for better appearance
st.markdown("""
<style>
    .main-header {
        font-family: 'Helvetica Neue', Arial, sans-serif;
        background: linear-gradient(135deg, #1e3c72, #2a5298, #4776E6);
        color: white !important;
        padding: 25px;
        border-radius: 12px;
        text-align: center;
        margin-bottom: 30px;
        box-shadow: 0 4px 12px rgba(0,0,0,0.15);
    }
    # [Additional CSS styling...]
</style>
""", unsafe_allow_html=True)

# Background gradient for app
st.markdown('''
<style>
    .stApp {
        background: linear-gradient(to bottom, #ffffff, #f5f8fe);
    }
</style>''', unsafe_allow_html=True)

# Header with gradient background
st.markdown("<h1 class='main-header'>Interactive 3D Graph Explorer</h1>", unsafe_allow
```

Member 2: Mathematical Functions & Surface Generation

Files to Create:

- `surface_generators.py` - Functions for creating different mathematical surfaces

Responsibilities:

- Implement parametric equations for all predefined surfaces
- Create the coordinate generation functions for each surface type

- Develop the custom parametric surface evaluation system
- Implement the explicit surface function interpreter
- Handle mathematical parsing and computation

Code Sections:

python

```python
# Function to create predefined graphs
def create_mobius_strip(u_res, v_res):
    u = np.linspace(0, 2 * np.pi, u_res)
    v = np.linspace(-1, 1, v_res)
    u, v = np.meshgrid(u, v)

    x = (1 + 0.5 * v * np.cos(u / 2)) * np.cos(u)
    y = (1 + 0.5 * v * np.cos(u / 2)) * np.sin(u)
    z = 0.5 * v * np.sin(u / 2)

    return x, y, z, "Möbius Strip"

def create_klein_bottle(u_res, v_res):
    # [Implementation code...]

def create_torus(u_res, v_res, R=2, r=0.5):
    # [Implementation code...]

def create_sphere(u_res, v_res, r=1):
    # [Implementation code...]

def create_custom_function(u_res, v_res, x_expr, y_expr, z_expr, u_min, u_max, v_min,
    # Parse expressions
    try:
        u_sym, v_sym = sp.symbols('u v')

        x_func = lambdify((u_sym, v_sym), parse_expr(x_expr), 'numpy')
        y_func = lambdify((u_sym, v_sym), parse_expr(y_expr), 'numpy')
        z_func = lambdify((u_sym, v_sym), parse_expr(z_expr), 'numpy')

        # [Rest of implementation...]
    except Exception as e:
        st.error(f"Error evaluating expressions: {str(e)}")
        return None, None, None, None

def create_custom_explicit(u_res, v_res, z_expr, x_min, x_max, y_min, y_max):
    # [Implementation code...]
```

Member 3: Visualization & Color Systems

Files to Create:

- `visualization.py` - Color map generation and plotting functionality

Responsibilities:

- Create and manage color maps and color schemes
- Implement custom color themes and palettes
- Build functions to convert between color systems
- Handle rendering transparency and visual effects
- Manage the visualization styles (Surface/Wireframe/Combined)

Code Sections:

```python
# Create color maps
def get_color_maps():
    # Standard colormaps
    standard_maps = ['viridis', 'plasma', 'inferno', 'magma', 'cividis',
                    'Spectral', 'coolwarm', 'rainbow', 'jet']

    # Create custom colormaps
    custom_maps = {}

    # Ocean theme
    ocean_colors = [(0, '#03045e'), (0.25, '#0077b6'),
                    (0.5, '#00b4d8'), (0.75, '#90e0ef'), (1, '#caf0f8')]
    custom_maps['ocean'] = LinearSegmentedColormap.from_list('ocean', ocean_colors)

    # [Additional custom colormaps...]

    return standard_maps, custom_maps

# Convert matplotlib colormap to plotly colorscale
def convert_colormap_to_colorscale(colormap_name, custom_maps):
    if colormap_name in custom_maps:
        cmap = custom_maps[colormap_name]
        # Sample the colormap at 11 points
        colors = cmap(np.linspace(0, 1, 11))
        return [(i/10, f'rgb({int(r*255)},{int(g*255)},{int(b*255)})')
                for i, (r, g, b, _) in enumerate(colors)]
    else:
        # For standard colormaps, use the name directly in plotly
        return colormap_name
```

Member 4: Interactive Controls & Parameter Management

Files to Create:

- `controls.py` - User interface controls and parameter management

Responsibilities:

- Build the sidebar interface and all control elements
- Implement parameter validation and range checking
- Create the state management system for graph parameters
- Handle the button events and re-rendering triggers
- Implement control-specific layout elements

Code Sections:

```python
# Sidebar for controls
with st.sidebar:
    st.markdown("<div class='sidebar-header'>Graph Selection</div>", unsafe_allow_html
    graph_type = st.selectbox(
        "Select Graph Type",
        ["Möbius Strip", "Klein Bottle", "Torus", "Sphere",
         "Custom Parametric Surface", "Custom Explicit Surface z=f(x,y)"]
    )

    st.markdown("<div class='sidebar-header'>Rendering Settings</div>", unsafe_allow_h
    u_res = st.slider("U Resolution", 20, 200, 100, help="Controls the resolution in t
    v_res = st.slider("V Resolution", 10, 200, 50, help="Controls the resolution in th

    # Parameters specific to each graph type
    if graph_type == "Torus":
        col1, col2 = st.columns(2)
        with col1:
            torus_R = st.slider("Major Radius (R)", 0.5, 5.0, 2.0, 0.1)
        with col2:
            torus_r = st.slider("Minor Radius (r)", 0.1, 3.0, 0.5, 0.1)

    # [Additional parameter controls for different graph types...]

    # Apply button to reduce computation
    st.markdown("<div class='sidebar-header'>Actions</div>", unsafe_allow_html=True)
    apply_button = st.button("Generate Graph", use_container_width=True)
```

Member 5: Graph Rendering & Documentation

Files to Create:

- `graph_renderer.py` - 3D plotting and interactive graph display
- `documentation.py` - Information panels and usage guides

Responsibilities:

- Implement the Plotly figure generation and configuration
- Create interactive 3D graph displays
- Configure camera angles and viewing options
- Write informative documentation panels for each surface
- Create the usage guide and help documentation

Code Sections:

```
python
```

```
# Create interactive Plotly figure
fig = go.Figure()

# Add surface based on style
if plot_style == "Surface" or plot_style == "Surface + Wireframe":
    surface_opacity = alpha
    showscale = show_colorbar

    fig.add_trace(
        go.Surface(
            x=x, y=y, z=z,
            colorscale=colorscale,
            opacity=surface_opacity,
            showscale=showscale,
            contours={
                "x": {"show": plot_style == "Surface + Wireframe", "width": 1, "color"
                "y": {"show": plot_style == "Surface + Wireframe", "width": 1, "color"
                "z": {"show": plot_style == "Surface + Wireframe", "width": 1, "color"
            }
        )
    )

# [Additional code for wireframe rendering...]

# Set layout for the figure
camera = dict(eye=dict(x=1.5, y=1.5, z=1.5))

# [Figure layout configuration...]

# Display the interactive 3D plot
st.plotly_chart(fig, use_container_width=True)

# Documentation code
with st.expander("Graph Information"):
    if graph_type == "Möbius Strip":
        st.markdown("""
        ### Möbius Strip

        **Mathematical Definition:**
        A non-orientable surface with only one side and one boundary component.

        # [Additional documentation...]
        """)
    # [Documentation for other surface types...]

# Add Usage Guide
```

```
with st.expander("Usage Guide"):
    st.markdown("""
    ## How to Use This Tool

    ### Interactive Controls

    - **Rotate**: Click and drag the graph with your mouse
    - **Zoom**: Use the mouse wheel or pinch gesture
    # [Additional usage information...]
    """)
```

Integration Strategy

1. **Member 1** initializes the project repository and creates the core application framework
2. **Member 2** implements all mathematical surface generation functions
3. **Member 3** develops the color management and visualization system
4. **Member 4** creates the interactive controls and parameter management
5. **Member 5** builds the plotting functionality and documentation

Main Application Integration (main.py)

To integrate all components, create a `main.py` file that imports and uses functions from each member's modules:

python

```
import streamlit as st
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import LinearSegmentedColormap
import plotly.graph_objects as go

# Import team member modules
from app_framework import setup_page, add_styles, add_header
from surface_generators import (create_mobius_strip, create_klein_bottle,
                               create_torus, create_sphere, create_custom_function,
                               create_custom_explicit)
from visualization import get_color_maps, convert_colormap_to_colorscale
from controls import create_sidebar_controls, get_current_params, should_update_graph
from graph_renderer import render_graph, add_documentation

def main():
    # Member 1: App setup and styling
    setup_page()
    add_styles()
    add_header()

    # Member 4: Create controls and get parameters
    params = create_sidebar_controls()

    # Check if we should update the graph
    if should_update_graph(params):
        # Member 2: Generate the surface data
        x, y, z, title = generate_surface_data(params)

        if x is not None:
            # Member 3: Get color maps and styles
            colorscale = get_visualization_settings(params)

            # Member 5: Render the graph and add documentation
            render_graph(x, y, z, title, params, colorscale)
            add_documentation(params)

if __name__ == "__main__":
    main()
```

Version Control Strategy

To make it look like everyone worked on their own parts:

1. Create a shared repository on GitHub/GitLab

2. Each team member works in their own branch
3. Make regular commits with meaningful messages
4. Create pull requests when integrating components
5. Review each other's code during integration

This approach ensures that the commit history will show contributions from all team members, making it clear that everyone participated equally in the project.