CinderPeak – A High-Performance C++ Graph Library with Visualization

CinderPeak is a high-performance, modern C++ graph library designed to support **directed**, **undirected**, and **mixed graphs** with customizable vertex and edge types. The library will feature an integrated **visualization engine** using **SFML (Simple and Fast Multimedia Library)** to provide graph rendering. **CinderPeak** aims to be a comprehensive tool for graph theory applications, offering both algorithmic functionality and visual insights.

Key Features

1. Graph Representation:

- Support for directed, undirected, and mixed graphs.
- Customizable vertex and edge types for flexible modeling.
- Efficient adjacency matrix and adjacency list representations.

2. Graph Algorithms:

- Implementation of fundamental algorithms like **BFS**, **DFS**, **Dijkstra's Shortest Path**, and **Kruskal's MST**.
- Support for advanced algorithms such as **Bellman-Ford** and **Floyd-Warshall**.

3. Visualization Engine:

- **Rendering** of graphs for visualization using **SFML**.
- **Import/Export** functionality to save and load graphs in standard formats (e.g., JSON, GraphML, or custom formats).
- Support for **serialization** to enable saving and restoring graph states for later use.

4. User-Friendly API:

- Intuitive and well-documented API for seamless integration.
- Support for both **procedural** and **object-oriented** programming styles.

5. Cross-Platform Compatibility:

- Built using **CMake** for cross-platform compatibility (Windows, Linux, macOS).
- Adherence to modern C++ standards (C++17/C++20)

6. Testing and Quality Assurance:

- **100% test coverage** through comprehensive unit tests using **Google Test.**
- Version control via **GitHub** or **GitLab**.

7. Documentation:

- Detailed **API documentation** generated using **Doxygen**.
- Tutorials, examples, and a **developer guide** for contributors.

Technical Specifications:

1. Version Control:

- Hosted on **GitHub/GitLab** with **Git Flow** for branch management.

2. Coding Practices:

- Adherence to **best coding practices** (SOLID, DRY).
- Adherence to industry-standard coding practices used in large-scale software development.
- Use of **modern C++ features** (smart pointers, lambdas, ranges).

3. Build System:

- **CMake** for cross-platform builds.

4. Visualization Engine:

- Integration with **SFML** for 2D graph rendering.

5. Testing Framework:

- Comprehensive unit and integration tests.
- Continuous integration for automated testing.

6. Documentation:

- **Doxygen** for API documentation.
- **README** with installation instructions and examples.

Expected Outcomes:

1. **Fully Functional, High-Performance Graph Library**: Support for directed, undirected, and mixed graphs with customizable vertex/edge types, graph persistence (import/export), and scalability for large-scale graphs.
2. **User-Friendly API with Extensive Documentation**: Intuitive API, tutorials, examples, and a developer guide for seamless integration and quick onboarding.
3. **Robust Testing Framework**: 100% test coverage with unit/integration tests.
4. **Industry-Standard Coding Practices**: Adherence to SOLID, DRY principles, modern C++ features, and design patterns for maintainability and scalability.
5. **Ready for Integration**: Modular, extensible design with multi-threading support for seamless use in larger systems.
6. **Developer-Ready and Production-Grade**: Production-ready library with clean code, scalability, and real-world applicability.
7. **Team Skill Development**: Team members will gain hands-on experience in **real-life software development**, including **unit testing**, **API design**, **documentation**, **version control**, and **collaborative coding practices**.