Структура программы:

My3DModelViewer/

├── CMakeLists.txt

├── main.cpp

├── model.h

├── model.cpp

├── viewer.h

├── viewer.cpp

├── modelviewer.h

├── modelviewer.cpp

├── mainwindow.h

└── mainwindow.cpp

В проекте My3DModelViewer код распределен по нескольким файлам для обеспечения модульности, читаемости и удобства поддержки. Ниже приведено краткое описание содержимого каждого файла и обоснование такого распределения.

1. CMakeLists.txt

• Содержание: Конфигурация сборки проекта с использованием CMake.

• Зачем: Указывает компилятору, как собирать проект, какие библиотеки подключать и какие файлы компилировать.

• Почему: CMake обеспечивает кроссплатформенность и упрощает сборку проекта.

2. main.cpp

• Содержание: Точка входа в программу. Создает объект приложения QApplication и главное окно MainWindow.

• Зачем: Инициализирует приложение и запускает основной цикл событий.

• Почему: Отделение точки входа от остального кода упрощает тестирование и расширение функциональности.

3. mainwindow.h и mainwindow.cpp

• Содержание: Класс MainWindow, который представляет главное окно приложения. Включает методы для открытия и сохранения моделей.

• Зачем: Управляет основным интерфейсом приложения и обработкой пользовательских действий (например, открытие файла).

• Почему: Отделение логики главного окна от остального кода делает программу более модульной и удобной для расширения.

4. model.h и model.cpp

• Содержание: Класс Model, который отвечает за загрузку и хранение 3D-модели, а также за расчет её параметров (объем, площадь проекции).

• Зачем: Инкапсулирует логику работы с 3D-моделью.

• Почему: Отделение логики модели от визуализации и интерфейса упрощает тестирование и повторное использование кода.

5. viewer.h и viewer.cpp

• Содержание: Класс Viewer, который отвечает за отображение 3D-модели и обработку пользовательских взаимодействий (вращение, масштабирование).

• Зачем: Управляет визуализацией модели и обработкой событий мыши.

• Почему: Отделение логики отображения от остального кода делает программу более гибкой и удобной для расширения.

6. modelviewer.h и modelviewer.cpp

• Содержание: Класс ModelViewer, который объединяет Model и Viewer, обеспечивая взаимодействие между ними.

• Зачем: Связывает логику модели и её визуализацию.

• Почему: Отделение связующего кода от логики модели и отображения упрощает поддержку и тестирование.

7. mainwindow.ui (если используется)

• Содержание: XML-файл, описывающий интерфейс главного окна (если используется Qt Designer).

• Зачем: Упрощает создание и изменение интерфейса.

• Почему: Отделение интерфейса от логики программы делает код более читаемым и удобным для изменения.

Итог:

Распределение кода по файлам сделано для обеспечения:

1. Модульности: Каждый класс отвечает за свою часть функциональности.

2. Читаемости: Код легче понимать и поддерживать.

3. Повторного использования: Классы можно использовать в других проектах.

4. Тестируемости: Каждый модуль можно тестировать отдельно.

Такая структура проекта делает его более организованным и удобным для дальнейшего развития.