Министерство образования и науки Украины  
Одесский Национальный Университет им. И.И. Мечникова

Институт Математики, Экономики и Механики

**Создание иерархии классов на тему «Геометрические фигуры: многогранники»**

выполнил: Кукишев А.А., студент I курса факультета Компьютерной Инженерии

руководитель:

канд. ф.

Одесса – 2016

Содержание

**1 Постановка задачи**

Необходимо создать классы: «Многогранник» (абстрактный) и его подклассы: «Пирамида» (характеризуется многоугольником в основании и высотой), «Призма» (характеризуется многоугольником в основании и высотой), «Додекаэдр» (характеризуется стороной). Создать абстрактный класс «Многоугольник» с наследниками: «Треугольник», «Прямоугольник», «Правильный многоугольник» (характеризуется количеством сторон и стороной). Объявить методы площадь, периметр, количество вершин для многоугольника и реализовать их в наследниках. Объявить методы площадь поверхности, объем, количество вершин, количество ребер и количество граней для многогранника и реализовать их в конкретном многограннике.

В проекте предусмотреть возможность создания новых многогранников произвольных типов и выдачи полных характеристик многогранников (в табличной форме и подробно для каждого многогранника). Для этого определить методы выдачи характеристик многогранника кратко (в одну строку) и подробно.

**2 Используемые средства**

2.1 Qt — кроссплатформенная библиотека разработки GUI на С++.

Библиотека Qt задумывалась и начиналась как кроссплатформенный тулкит (toolkit) для быстрой разработки графических интерфейсов (GUI) приложений на языке C++, с целью упростить жизнь программистов, пишущих на C++ кроссплатформенные, переносимые GUI-приложения, которые должны работать и в среде Windows, Symbian, и в среде Unix/Linux под X11, и на компьютерах Macintosh. Qt является полностью объектно-ориентированным, легко расширяемым и поддерживающим технику компонентного программирования.

В настоящее время Qt значительно переросла рамки тулкита для разработки графических интерфейсов приложений. Она предоставляет использующему её программисту целостный фреймворк (framework), позволяющий при написании большей части приложения использовать только «родные» классы Qt и практически полностью отказаться от написания системно-зависимого кода. В библиотеке предусмотрены классы и для работы со строками, и для работы с файлами, сетью, базами данных, XML, и для обеспечения многопоточности в приложении, и многое другое.

Отличительная особенность Qt от других библиотек — использование Meta Object Compiler (MOC) — предварительной системы обработки исходного кода. MOC позволяет во много раз увеличить мощь библиотек, вводя такие понятия, как слоты и сигналы. Кроме того, это позволяет сделать код более лаконичным. Утилита MOC ищет в заголовочных файлах на C++ описания классов, содержащие макрос Q\_OBJECT, и создаёт дополнительный исходный файл на C++, содержащий метаобъектный код.

Qt позволяет создавать собственные плагины и размещать их непосредственно в панели визуального редактора. Также существует возможность расширения привычной функциональности виджетов, связанной с размещением их на экране, отображением, перерисовкой при изменении размеров окна.

Qt комплектуется визуальной средой разработки графического интерфейса «Qt Designer», позволяющей создавать диалоги и формы в режиме WYSIWYG. В поставке Qt есть «Qt Linguist» — графическая утилита, позволяющая упростить локализацию и перевод программы на многие языки; и «Qt Assistant» — справочная система Qt, упрощающая работу с документацией по библиотеке, а также позволяющая создавать кроссплатформенную справку для разрабатываемого на основе Qt ПО. Начиная с версии 4.5.0 в комплект Qt включена среда разработки «Qt Creator», которая включает в себя редактор кода, справку, графические средства «Qt Designer» и возможность отладки приложений. «Qt Creator» может использовать GCC или Microsoft VC++ в качестве компилятора и GDB в качестве отладчика. Для Windows версий библиотека комплектуется компилятором, заголовочными и объектными файлами MinGW.

2.2 OpenGL Mathematics

GLM (OpenGL Mathematics — математика для OpenGL) — библиотека для OpenGL, предоставляющая программисту на C++ структуры и функции, позволяющие использовать данные для OpenGL.

Одна из особенностей GLM состоит в том, что его реализация основана на спецификации GLSL (OpenGL Shading Language).

**3. Теоретическая мат.**

**3.1 Обзор предметной области.**

**3.2 Обзор принципов ООП.**

Объектно-ориентированное программирование (ООП) — методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования.

Наследование — один из четырёх важнейших механизмов объектно-ориентированного программирования (наряду с инкапсуляцией, полиморфизмом и абстракцией), позволяющий описать новый класс на основе уже существующего (родительского), при этом свойства и функциональность родительского класса заимствуются новым классом. Другими словами, класс-наследник реализует спецификацию уже существующего класса (базовый класс). Это позволяет обращаться с объектами класса-наследника точно так же, как с объектами базового класса. Простое наследование: Класс, от которого произошло наследование, называется базовым или родительским (англ. base class). Классы, которые произошли от базового, называются потомками, наследниками или производными классами (англ. derived class). В некоторых языках используются абстрактные классы. Абстрактный класс — это класс, содержащий хотя бы один абстрактный метод, он описан в программе, имеет поля, методы и не может использоваться для непосредственного создания объекта. То есть от абстрактного класса можно только наследовать. Объекты создаются только на основе производных классов, наследованных от абстрактного.

Полиморфизм — один из четырёх важнейших механизмов объектно-ориентированного программирования (наряду с абстракцией, инкапсуляцией и наследованием). Полиморфизм позволяет писать более абстрактные программы и повысить коэффициент повторного использования кода. Общие свойства объектов объединяются в систему, которую могут называть по-разному — интерфейс, класс. Общность имеет внешнее и внутреннее выражение: внешняя общность проявляется как одинаковый набор методов с одинаковыми именами и сигнатурами (именем методов и типами аргументов, и их количеством); внутренняя общность — одинаковая функциональность методов. Её можно описать интуитивно или выразить в виде строгих законов, правил, которым должны подчиняться методы. Возможность приписывать разную функциональность одному методу (функции, операции) называется перегрузкой метода (перегрузкой функций, перегрузкой операций).

**3. Программная реализация**

3.1 Диаграмма классов геометрических фигур.

На рисунке 3.1 представлена диаграмма системы классов геометрических фигур.

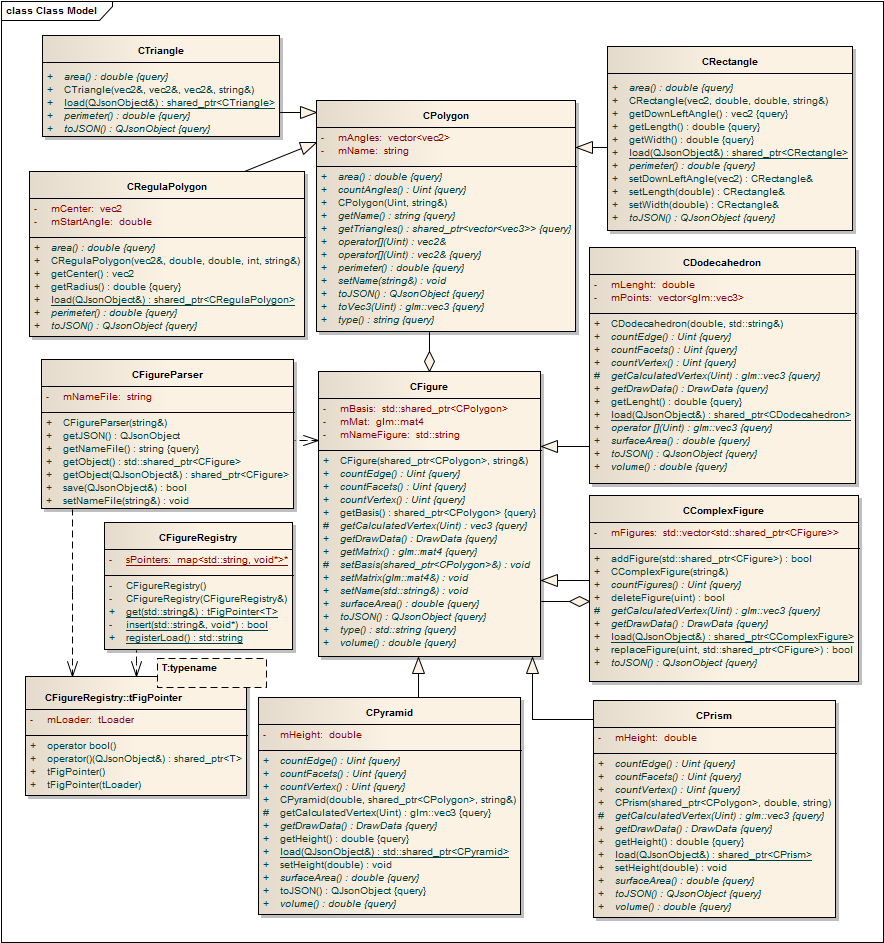


Рисунок 3.1. Диаграмма классов геометрических фигур.

На диаграмме представлены следующие классы:

* **Класс CPolygon.**

Класс CPolygon, определяет абстрактный многоугольник; задается количеством углов и именем фигуры.

Класс CPolygon имеет следующие поля:

std::vector<glm::vec2> mAngles – вектор вершин многоугольника (углов многоугольника).

std::string mName - имя фигуры.

Основные публичные методы класса CPolygon:

virtual QJsonObject toJSON() const = 0; - метод, преобразовывающий объекты потомков в json-объект, который используется для записи в файл формата json.

virtual const string type() const; - метод, возвращающий строковое значение имени класса фигуры.

virtual glm::vec3 toVec3(Uint index) const; - метод, преобразовывающий двухмерную вершину полигона в трехмерную вершину. index – индекс вершины.

virtual shared\_ptr<vector<vec3>> getTriangles() const; - метод, предназначенный для триангуляции многоугольника.

virtual double area() const = 0; - метод возвращает площадь полигона.

virtual double perimeter () const = 0; - метод возвращает периметр полигона.

* **Класс CFigure.**

Класс CFigure определяет абстрактную трехмерную фигуру. Задается основой (CPolygon) и именем.

Основные поля класса:

std::string mNameFigure; - имя фигуры.

glm::mat4 mMat; - матрица трансформирования фигуры.

std::shared\_ptr<CPolygon> mBasis; - основа трехмерной фигуры.

Основные публичные методы:

virtual double surfaceArea() const = 0; - метод возвращает площадь поверхности фигуры.

virtual double volume() const = 0; - метод возвращает объём фигуры.

virtual QJsonObject toJSON() const = 0; - метод, преобразовывающий объекты потомков в json-объект, который используется для записи в файл формата json.

virtual const std::string type() const; - метод, возвращающий строковое значение имени класса фигуры.

vec3 getCalculatedVertex(Uint index) const = 0; - защищенный метод для вычисления вершин наследников.

virtual DrawData getDrawData() const = 0; - метод выполняет триангуляцию трехмерной фигуры. Рассчитанные данные используются для отрисовки фигуры.

* **Класс DrawData.**

Класс DrawData содержит множество последовательных вершин, представляющие собой треугольники, полученные в процессе триангуляции наследников CFigure, а также упорядоченное множество цветов для каждой вершины.

* **Класс CFigureRegistry.**

Данный класс предназначен для регистрации статического метода load потомков класса CFigure. Это необходимо для разрешения проблемы идентификации типа фигуры при загрузке из json-формата. Регистрация метода load осуществляется путем сопоставления RTTI-имени класса и указателя на метод load.

Для удобства регистрации фигуры были написаны два макроса: DECLARE\_FIGURE и REGISTER\_FIGURE(\_\_class\_name ).

Макрос DECLARE\_FIGURE дополняет класс дополнительным статическим полем(\_\_static\_type\_name\_\_), хранящий в себе RTTI-имя класса, и виртуальным методов type(), возвращающий константную ссылку на статическое поле.

Пример использования данного макроса:

class CPrism: public CFigure

{

DECLARE\_FIGURE

…

}

Макрос REGISTER\_FIGURE(\_\_class\_name ) используется для регистрации метода load фигуры, а также инициализации статического поля \_\_static\_type\_name\_\_ . Данный макрос следует использовать в файле реализации трехмерной фигуры (cpp).

Пример использования данного макроса:

#include "cprism.h"

REGISTER\_FIGURE(CPrism)

Для хранения регистрационных данных о метода load, класс CFigureRegister содержит в себе приватное статическое поле std::map<std::string, void\*> \*sPointers.

Для регистрации данных и доступа к ним служат два статических шаблонных метода:

static std::string registerLoad();

static tFigPointer<T> get(const std::string& name);

Пример использования класса CFigureRegister:

Пусть есть объект типа QJsonObject obj, в котором есть все необходимые значения, в том числе и тип фигуры, для создания какого-то объекта наследника класса CFigure. Какая фигура конкретно находится в объекте – неизвестно. Но с помощью класса CFigureRegister можно вызвать метод load, который выполнит преобразование данного объект в конкретного наследника CFigure:

std::shared\_ptr<CFigure> figure = CFigureRegistry::get<CFigure>

( obj.find("type").value().toString().toStdString() )( obj );

Статический метод load имеют все наследники классов CFigure и CPolygon.

* **Класс CRectangle.**

Класс CRectangle является наследником CPolygon и определяет прямоугольник, который задается координатам левого нижнего угла, шириной, длиной и именем фигуры.

* **Класс CTriangle.**

Класс CTriangle является наследником CPolygon и определяет треугольник; задается тремя точками треугольника и именем.

* **Класс CRegularPolygon.**

Класс CRegularPolygon является наследником CPolygon и определяет правильный многоугольник. Задается координатами центра многоугольника, начальным углом, радиусом, количеством углов правильного многоугольника и именем.

* **Класс CPyramid.**

Класс CPyramid является наследником класса CFigure и представляет собой пирамиду. Объекты CPyramid задаются основой, высотой и именем.

* **Класс CPrism.**

Класс CPrism является наследником класса CFigure и представляет собой призму. Объекты CPrism задаются основой, высотой и именем.

* **Класс CDodecahedron.**

Класс CDodecahedron является наследником CFigure и представляет собой додекаэдр. Объекты CDodecahedron задаются длиной ребра додекаэдра и именем.

* **Класс CComplexFigure.**

Класс CComplexFigure является наследником CFigure и представляет собой список фигур, или сложную фигуру, которая состоит из множества фигур. Этот класс содержит в себе вектор умных указателей на фигуры и в нем определены такие методы, как: добавить фигуру, удалить фигуру, заменить фигуру, метод преобразования в json-представление и метод загрузки с json-представления.

* **Класс CFigureParser.**

Основное предназначение этого класс: сохранять обьекты классов геометрических фигур в json файлы, преобразовывать json-представление объекта фигуры в указатель на объект фигуры.

Исходный код вышеописанных классов приведет в приложении А.

3.2 Диаграмма классов графического интерфейса пользователя.

На рисунке 3.2 представлена диаграмма системы классов графического интерфейса пользователя.

* **Класс MainWindow.**

Класс MainWindow является главным классом системы интерфейса пользователя. В нем осуществляется загрузка (сохранение) списка объектов наследников класса CFigure с файлов (в файл) формата json, добавление и удаление фигуры в список, редактирование фигуры в списке и её отображение.

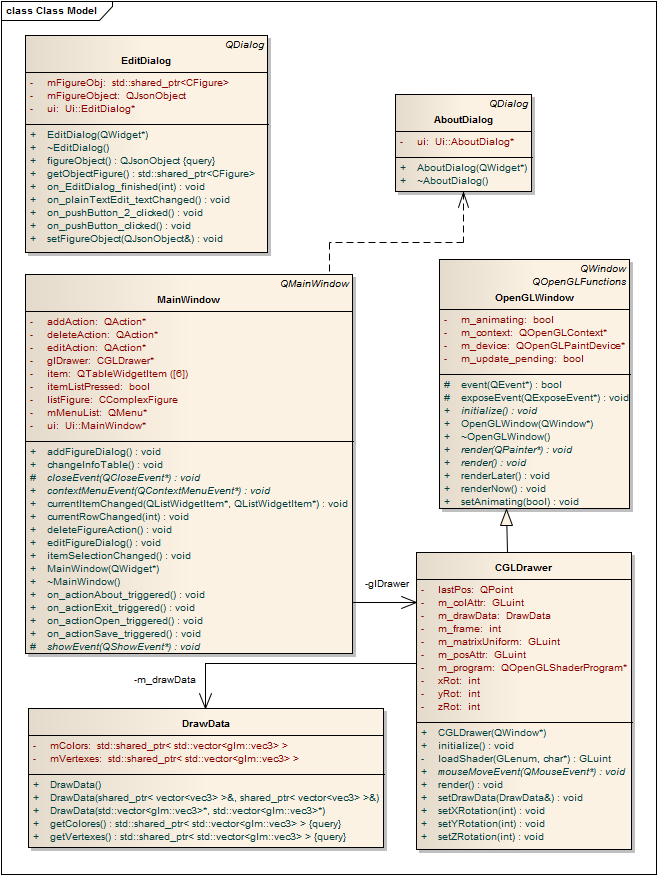


Рисунок 3.2. Диаграмма классов графического интерфейса пользователя.

Основные слоты класса MainWindow:

Слот - это функция, которая вызывается в ответ на определенный сигнал. Сигнал вырабатывается, когда происходит определенное событие (нажатие кнопки в программе и т.п.).

void editFigureDialog(); - слот, который вызывает окно редактирования фигуры из списка и вносит изменение в список после окончания работы с окном редактирования.

void addFigureDialog(); - слот, который вызывает окно добавление фигуры в список и вносит соответствующие изменения в список фигур.

Основные поля класса CFigure:

CComplexFigure listFigure; - список фигур.

CGLDrawer \*glDrawer; - окно отображения фигур.

* **Класс EditDialog.**

Класс EditDialog предназначен для редактирования фигуры в списке и добавление новых фигур в списке.

Основные поля класса EditDialog:

QJsonObject mFigureObject; - json-представление обьекта трехмерной фигуры.

std::shared\_ptr<CFigure> mFigureObj; - фигура, которая хранится в процессе работы окна EditDialog.

Основные методы класса EditDialog:

QJsonObject figureObject() const; - возвращает mFigureObject;

void setFigureObject(const QJsonObject &figureObject); - сеттер для mFigureObject.

shared\_ptr <CFigure> getObjectFigure(); - возвращает mFigureObj.

* **Класс OpenGLWindow и CGLDrawer.**

Класс CGLDrawer осуществляет отрисовку наследников CFigure. Он является наследником класса, который был взят из примеров QT работы с OpenGL. Пример можно посмотреть по следующей ссылке:

http://doc.qt.io/qt-5/qtgui-openglwindow-example.html

DrawData m\_drawData; - упорядоченное множество вершин, представимые треугольниками, и множество цветов каждой вершины.

QOpenGLShaderProgram \*m\_program; - шейдерная программа.

Основные методы класса CGLDrawer:

void initialize() Q\_DECL\_OVERRIDE; - переопределенный метод класса OpenGLWindow, который инициализирует m\_program.

void render() Q\_DECL\_OVERRIDE; - переопределенный метод класса OpenGLWindow, в котором осуществляется отрисовка сцены.

Исходный код вышеописанных классов приведет в приложении В.