Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мультимедійних систем факультету інформатики

Графічна система для навчання

Текстова частина до курсової роботи

за спеціальністю “Комп’ютерні Науки” 122

Керівник курсової роботи

к.ф.-м.с., доцент Жежерун О. П.

“____” _____ 2020 р.

Виконав студент ФІ-МП1

Димченко О. В.

“____” _____ 2020 р.

ЗМІСТ

Анотація

Робота присвячена створенню системи для малювання геометрії в 2D, опису усіх використаних технологій, задач з якими зіштовхнувся та їх вирішеннями. Роботу виконав студент 1-го курсу магістратури факультету Інформатики Димченко Олексій, науковий керівник доцент Жежерун Олександр Петрович.

ВСТУП

Системи для малювання геометричних задач в 2D – це додаток за допомогою якого ви можете малювати примітиви потипу трикутників, кіл, прямокутників, ліній, трапецій, точок, кутів і подібного. А також виконувати усілякі функції над цими примітивами, наприклад видаляти їх, або проводити висоту і т.д.. Тим самим будуючи графік деякої абстрактної задачі, дані якої будуть зберігатись і оброблятись системою.

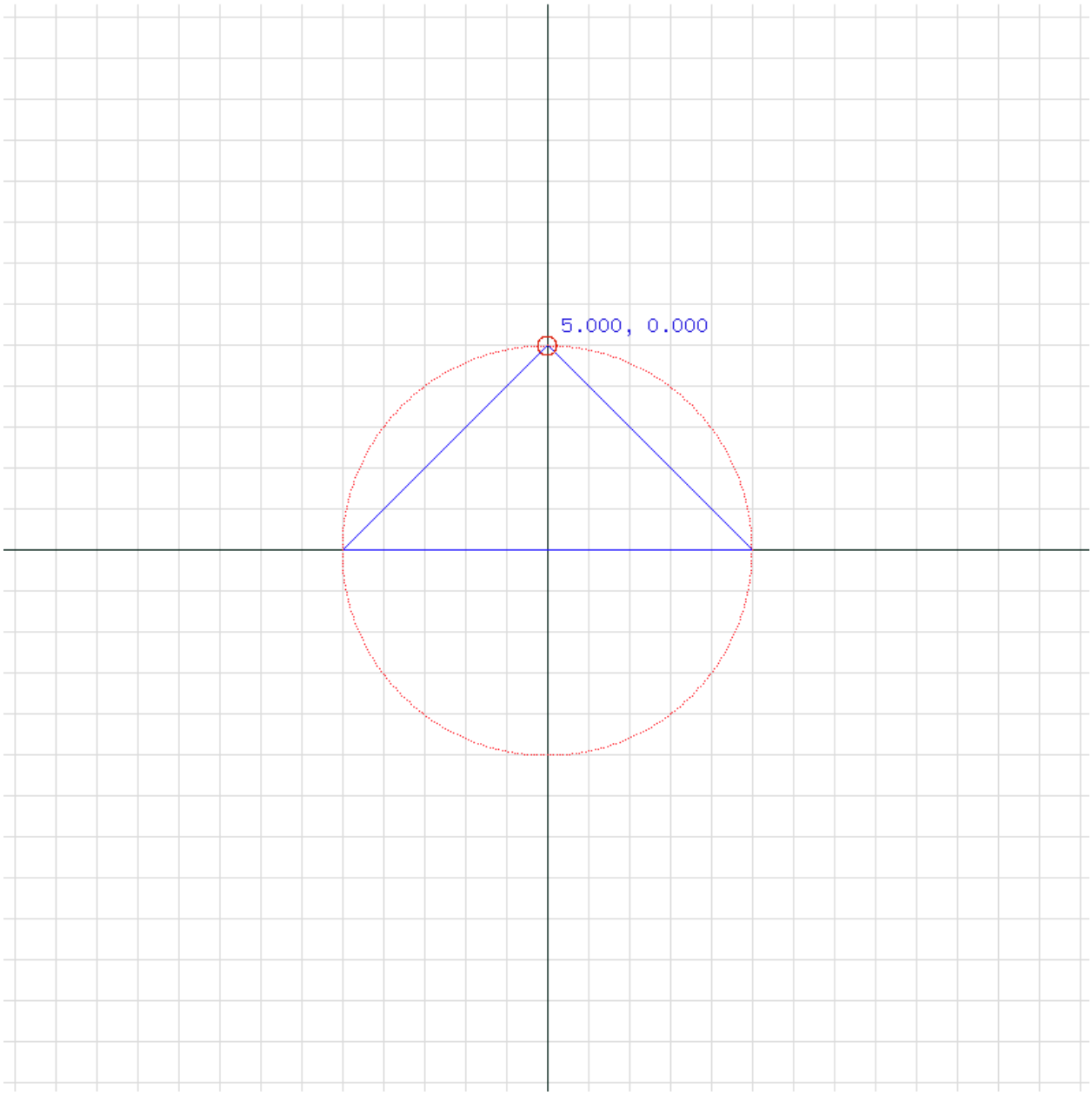
Метою написання системи для малювання геометричних задач в 2D є наступні пункти

1) Порівняти, обрати та навчитись новим (для мене) рендер бібліотекам в процесі створення системи
2) Розробити архітектуру застосунку, як шаблон для інших подібних додатків
3) Розробити застосунок який легко розширювати: додавати нові об’єкти для малювання, та нові інструменти для малювання тощо.
4) Розвити додаток в навчальний для навчання

1. Система малювання геометрії в 2D

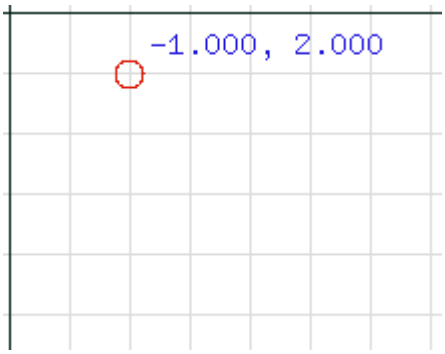
1.1 Опис системи

Система має складатись з двох частин. Перша це відображення того, що ми малюємо, тобто графік, інша це інтерфейс користувача. Приклад (Мал. 1) графік.



Мал.1

В программі є сітка на перетині якої знаходяться точки (потенційні координати) по яким користувач може малювати. (Мал.2)

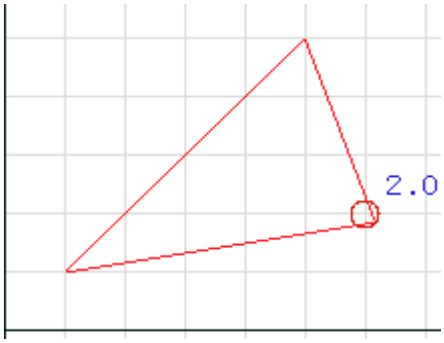


Мал.2

За допомогою правої кнопки миші можна відкрити меню (Мал.3)

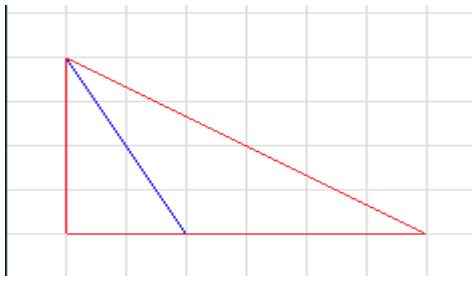
Мал.3

В меню можна обрати інструменти для малювання. Існуючі інструменти досить схожі в тому як їх застосовувати. Користувач обирає точки та по обраним точкам створюється відповідний об'єкт.



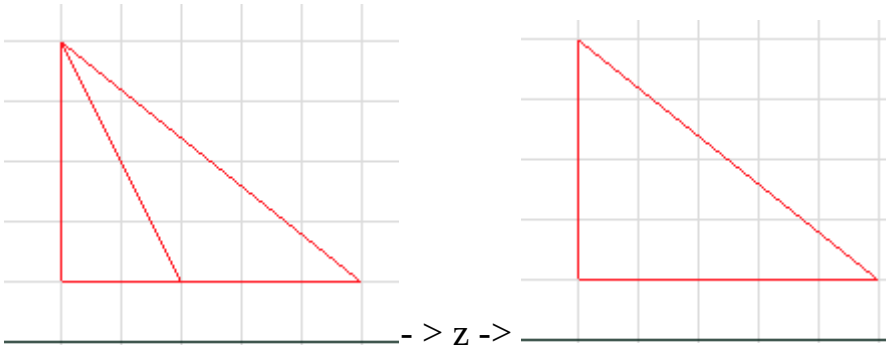
Мал.4

Також є можливість виділяти об'єкти та видаляти їх натискаючи ‘d’.



Мал.5

(Мал. 5) Виділений об’єкт. Якщо ви випадко видалели щось не те, або створили не правильний об’єкт в системі присутня можливість повертати дії назад, по типу Ctrl-Z та робити дії заново типу Ctrl-Y при натисканні ‘z’ та ‘y’ відповідно (Мал. 6)



Мал. 6

2. Опис використаних технологій

2.1 C++

Мова програмування C++ є потужним інструментом розробки низькорівневих програм. Головною перевагою C++ є насамперед швидкість виконання, яка досягається завдяки володіння ресурсами обчислювальної машини на найнижчому рівні і багатьом аспектам мови що допомагають у керуванні цими ресурсами. C++ підходить для написання додатків з використанням 2D та 3D графіки, оскільки є дуже швидким у виконанні та з його допомогою дуже зручно керувати ресурсами, буферами для вершин індексів, кольорів, тощо., а також для цієї мови існують багато різних бібліотек для малювання напряму спілкуючись з відеокартою, які потребують точності від програмістів.

Чому обрав C++ ?

1. Маю досвід роботи з ним
2. Підходить для графічних застосувань
3. Підходить для розширення та метапрограмування

2.2 OpenGL

OpenGL - специфікація, що визначає крос-платформний програмний інтерфейс, який не залежить від мови або середовища для створення 2D та 3D графіки.

Включає більше 250-ти функцій для рендеру 2D та 3D графіки. Використовується при створенні відеоігор, додатків для малювання, віртуальної реальності, тощо.

Є основним конкурентом з DirectX. OpenGL стоїть на двох стовпах:

1. Приховати деталі реалізації рендеру за допомогою відеокарт надаючи розробнику єдиний API
2. Приховати відмінності для різних платформ (Windows, Linux...)

Основним принципом роботи OpenGL є отримання наборів векторних графічних примітивів у вигляді точок, ліній і багатокутників з наступною математичною обробкою отриманих даних і побудовою растрового зображення.

Векторні трансформації і растеризація виконуються по graphics pipeline’у, який, по суті, являє собою дискретний автомат.

Чому обрав OpenGL ?

1. Простіший ніж DirectX.
2. Open source (має багато додаткових бібліотек)
3. Гідний конкурент DirectX.
4. Має хорошу документацію
5. Сам хотів розібратись з ним

2.3 Glut

Бібліотека OpenGL не використовує конкретної системи для управління вінками та девайсами. В результаті бібліотека OpenGL - це просто інтерфейс для 2D рендеру та 3D рендеру.

Glut - конкретна система управління вікнами, яка сама подбає про відкриття, закриття, зміну зображення та відображення вікон, а також обробить сигнали з девайсів. Бібліотека вирішила собою багато проблемних моментів, які були в бібліотеці OpenGL.

GLUT - це єдиний інтерфейс для роботи з вікнами, клавіатурою та мишкою (девайсами) незалежно від платформи. Додатки що використовують Glut можуть бути з легкістю перенесені з платформи на платформу, майже без переписування вихідного коду. Також GLUT дуже полегшує написання коду для додатків, які використовують бібліотеку OpenGL. Бібліотека Glut невелика, зручна, повноцінна, а також має чудову документацію. Програмний інтерфейс бібліотеки Glut являє собою «state machine». Це означає, що GLUT містить ряд змінних стану, які змінюються під час виконання програми.

Початкові стани машини Glut були обрані з розрахунком задовольняти більшість додатків. Програма може змінювати значення цих змінних стану, для задоволення специфічних вимог. Кожен раз при виконанні функції Glut змінюють свою поведінку залежно від значень змінних стану. Функції Glut прості і вимагають мінімуму параметрів.

Чому обрав саме Glut ?

1. Потрібно десь відображати зображення та обробляти інформацію отриману за її допомогою
2. Glut підтримує OpenGL

3. Має хорошу документацію
4. Простий у використанні
5. Задовільняє усі потреби для розробки системи

3. Архітектура

3.1 Data

Дані знаходяться в об’єкті Data, що являє собою Singletone - шаблон проектування, який гарантує, що клас матиме тільки один екземпляр, і забезпечує глобальну точку доступу до цього екземпляра.. Чому саме singletone ? Тому що непогано мати уніфікований доступ для читання даних з будь-якої точки програми, хто знає, де і які данні нам знадобляться. Також, якщо розширювати систему можна додати базу даних і тоді клас Data претвориться на клас який відповідає за з'єднання та роботу з базою даних буде статичним конектором до бази даних, при цьому буде мінімальна зміна коду, адже класи які відповідають за з’єднання часто є singletone’ами.

```
struct Data
{
    Data(const Screen& s, const Camera& camera) { ... }

    Grid grid{ (float)width, (float)height, horStep, verStep };

    Screen screen;
    Camera camera;
    SnapPoints snapPoints;
    DrawingObjects objects;
    Transactions transactions;

    std::unique_ptr<BaseTool> drawingTool;

    static Data data;

    static Data & Get()
    {
        return data;
    }
};
```

Рис 1.

Дані включають в себе:

1. Інформація про екран, положення мишки, ширига та висота вікон
2. Інформація про камеру (як відображати об’єкти)
3. Інформація про точки (SnapPoint) які по суті є світовими кординатами

4. Інформація про об'єкти які можуть бути відображені
5. Стек транзакцій
6. Інша інформація

Об’єкти які можуть бути намальовані зберігаються в класі DrawingObjects, який являє собою патерн Storage

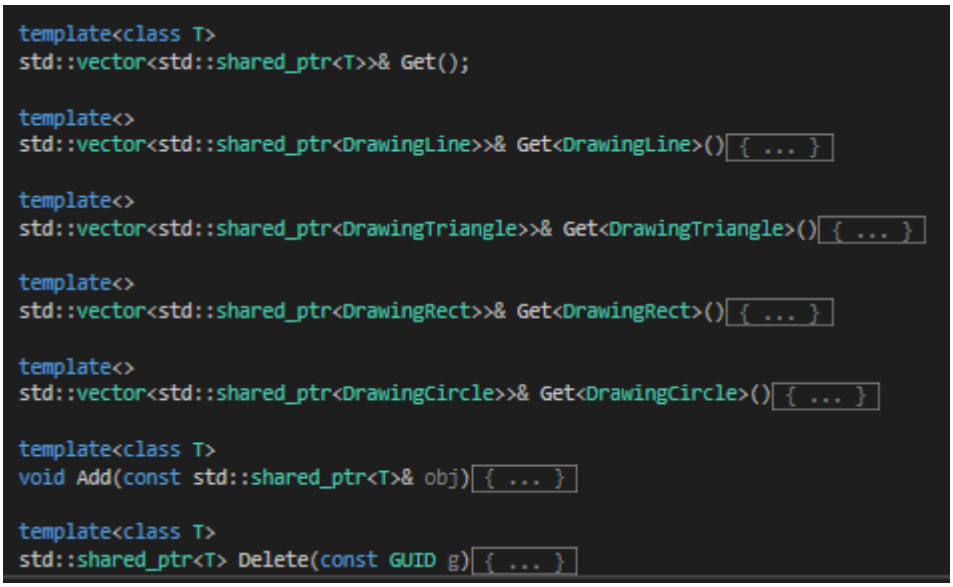


Рис 2.

Чому саме Storage ?

Теперь ми можемо легко розширювати цей клас та додавати будь-які нові об'єкти до системи. Тепер все що нам потрібно, щоб додати та маніпулювати цими об'єктами це додати метод Get<T> та параметризувати його нашим типом. Після чого усі методи які відповідають за обробку даних автоматично будуть генерувати функції для роботи з новим доданим типом. А саме Add, Delete, Change і т.д.

3.2 Менеджери

В програмі присутні так звані менеджери. Менеджери це класи які відповідають за зміну даних, проте вони самі не мають станів, а тому ці класи є статичними (в певному розумінні). По суті менеджери це імплементація патерну Mediator, але без станів. Менеджери відповідають за усі зміни з даними або з станом системи, тобто через них будь-які зміни відбуваються. Це приводить до того, що, інші класи не залежать від даних, а залежать тільки від інтерфейсу менеджерів. Також рівень менеджерів допомагає логуванню. Адже будь-яка зміна даних буде проходити через функції менеджеру, в яких можна викликати функції логування. Як приклад розглянемо DrawingManager.

```
struct DrawingManager
{
public:
    template<class T>
    static void Draw(const T&);

    template<>
    static void Draw<Line>(const Line& line) { ... }


    template<>
    static void Draw<Grid>(const Grid& grid) { ... }

    template<>
    static void Draw<SnapPoint>(const SnapPoint& p) { ... }


    template<>
    static void Draw<DrawingTriangle>(const DrawingTriangle& dt) { ... }

    template<>
    static void Draw<DrawingRect>(const DrawingRect& dr) { ... }
```

Рис 3.

Клас DrawingManager має шаблон статичного методу Draw, таким чином після додавання нового об’єкту, який можна малювати, все що нам потрібно, це додати нову конкретизацію шаблона та імплементувати її. Оскільки будь-яке малювання відбувається через DrawingManager, ми можемо легко досліджувати проблеми та оптимізувати код.

3.3 Об’єкти

Об’єкти які можна малювати в системі називаються з приставкою Drawing_ - це класи які наслідуються від базового класу для малювання. І мають ряд особливостей.

1. Вони підпорядковуються бізнес логіці програми (тобто по суті зберігають інформацію про трикутники, лінії, вершини і т.д.)
2. Вони мають унікальний Id (GUID)
3. Їх можна малювати на екрані
4. Вони зберігаються в даних
5. Вони знають як перевірити чи мишка їх вибрала (RayIntersection)

Як приклад DrawingTriangle. Хочу також звернути увагу, що ці класи досить прості і не мають майже ніяких імплементацій, тобто несуть в собі майже лише інформацію, а не поведінку.

```
struct DrawingTriangle : public DrawingObject
{
    DrawingTriangle(const Point& p1, const Point& p2, const Point& p3);

    virtual bool intersect(const Point&) override;

    std::array<Point, 3> points;
};

bool PointInTriangle(const Point& pt, const Point& v1, const Point& v2, const Point& v3);
```

Рис 4.

Проблемою може слугувати лише їх створення. Адже потрібно перевірити багато різних предикатів. Наприклад трикутник не може бути з двома однаковими вершинами і т.д.

3.4 Інструменти малювання

В системі присутні інструменти для малювання ліній, кругів, трикутників, прямокутників і т.д. Ці інструменти наслідуються від базового інструмента, який має особливий інтерфейс. Цей інтерфейс передається в Glut State Machine.

```
struct BaseTool
{
    BaseTool(const DrawingToolType type) : type(type) { }

    virtual ~BaseTool() { }
    virtual void mouseFunc(int button, int state, int x, int y) = 0;
    virtual void keyboardFunc(unsigned char key, int x, int y) = 0;
    virtual void passiveMotionFunc(int x, int y) = 0;

    DrawingToolType type;
};
```

Рис 6.

Тобто обробка натискання клавіш клавіатури, мишки, а також пересування мишки та будь які інші дії з вікнами тепер напряму викликають функції обраного інструмента для малювання.

```
void MouseManager::mouseFunc(int button, int state, int x, int y)
{
    //auto p = Point{ (float)x, (float)y + (fabs((float)Data::Get(
    if (Data::Get().drawingTool != nullptr)
    {
        Data::Get().drawingTool->mouseFunc(button, state, x, y);
    }
```

Рис 7.

Для прикладу приведу MouseManager, який відповідає за будь-яку взаємодію із мишкою. Якщо якийсь-інструмент обраний, то MouseManager викличе аналогічну функції через інтерфейс базового класу інструмента. По суті це патерн State, тому що в залежності від обраного нами варіанту ми змінюємо поведінку користування мишкою, клавіатурою, тощо.

Інструменти можуть зберігати в собі додаткові (проміжні) дані під час їх виконання. Ціль у інструмента одна – змінити дані для малювання. Проте

будь-який інструмент робить це не на пряму, а через TransactionManager. Тобто по суті сам інструмент може тільки читати дані, а змінює їх через конкретний TransactionManager створюючи відповідну транзакцію.

3.5 Транзації

Транзакція – це інтерфейс який має два методи undo та redo. Транзакція передбачає, що завдання яке вона несе в собі буде виконане в конструкторі, а в деструкторі транзакція просто видалить дані яка вона утримує. Це напів ідіома Resource Acquisition Is Initialization (RAII), чому я використав саме її далі.

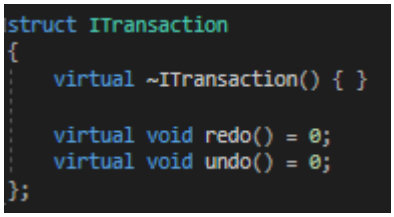


Рис 8.

Для прикладу розглянемо TransactionCreate. Це клас шаблон який наслідується від базового інтерфейсу транзакції та імплементує її методи. Ця транзакція потребує параметром шаблону DrawingObject, тобто об’єкт який можна намалювати. В конструктор на вхід отримує будь які аргументи за умови що з цих аргументів можна зробити об’єкт класу який ми передали в параметр шаблону. Саме тут усі об’єкти створюються і Resource Acquisition саме тут і відбувається. Транзакція має в собі спільний указник на об'єкт. Тобто об'єкт що був створений буде зберігатись в даних, проте якщо його видалити з даних він все ще буде існувати в контексті транзакції. Таким чином можна реалізувати відміну дії або повтор дії. Навіть якщо ми видалили

об’єкт і натисли Ctrl+Z новий об’єкт не буде створено, оскільки поки існує транзакція об’єкт не буде видаленим, ми просто повернемо його в видимість даних.

```
template<class T>  <T>
struct TransactionCreate : public ITransaction
{
    template<class... Args>
    TransactionCreate(Args&&... args)
    {
        Data::Get().objects.Add<T>(std::make_shared<T>(std::forward<Args>(args)...));
        tObj = Data::Get().objects.Get<T>().back();
    }

    virtual void redo() override
    {
        Data::Get().objects.Add<T>(tObj);
    }

    virtual void undo() override
    {

        Data::Get().objects.Delete<T>(tObj->id);
    }

    std::shared_ptr<T> tObj;
};
```

Рис 9.

Усі транзакції створюються через статичний метод TransactionManager. В даних зберігаються два stack’а транзакцій один для undo, інший для redo

```
struct Transactions
{
    std::stack<std::shared_ptr<ITransaction>> undo;
    std::stack<std::shared_ptr<ITransaction>> redo;
};
```

Рис 10.

Після створення транзакції вона потрапляє в stack undo. Якщо ми захочемо повернути дію то ми викличемо undo у верхньої транзакції в стеці після чого запушимо цю транзакцію в стек redo і потім викинемо цю транзакцію с undo.

```
struct TransactionManager
{
    template<class T, class... Args>
    static void makeTransaction(Args&&... args)
    {
        Data::Get().transactions.undo.emplace(std::make_shared<T>(std::forward<Args>(args)...));
    }

    static void undoTransaction()
    {
        if (size(Data::Get().transactions.undo) != 0)
        {
            Data::Get().transactions.undo.top()->undo();
            Data::Get().transactions.redo.push(Data::Get().transactions.undo.top());
            Data::Get().transactions.undo.pop();
        }
    }

    static void redoTransaction()
    {
        if (size(Data::Get().transactions.redo) != 0)
        {
            Data::Get().transactions.redo.top()->redo();
            Data::Get().transactions.undo.push(Data::Get().transactions.redo.top());
            Data::Get().transactions.redo.pop();
        }
    }
};
```

Рис 10.

Повністю діаграма виглядає приблизно так (Рис 11.). Нижче об’єкти які можна малювати і можливі іструменти Як можна побачити Менеджери не залежать від даних вони лише змінюють їх (Рис 12.)

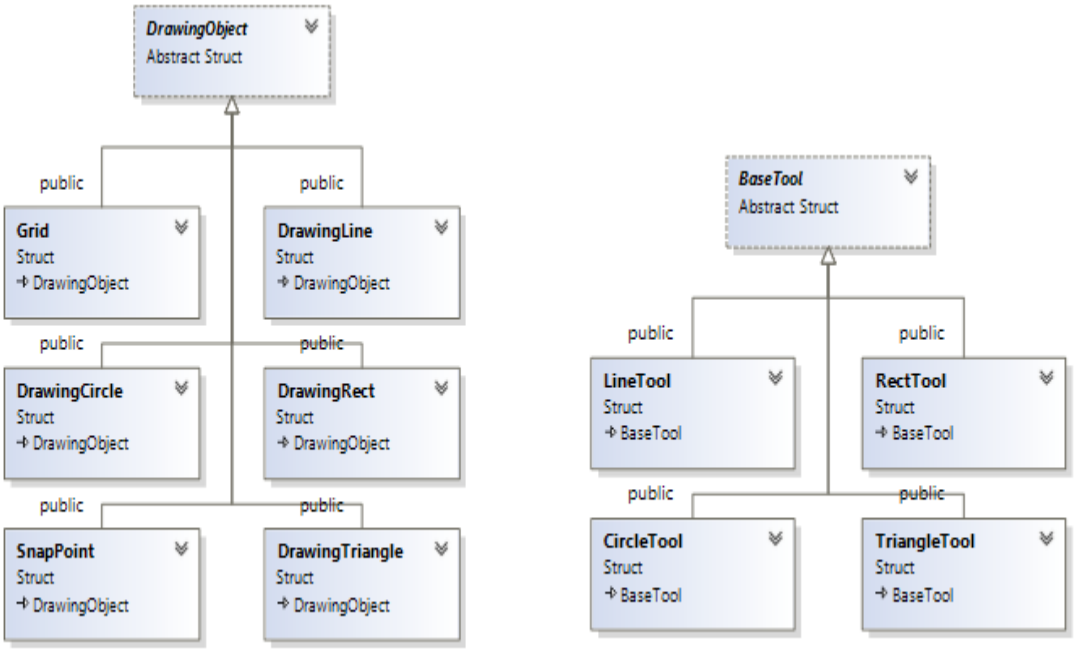

DrawingObject
Abstract Struct
public
public
Grid
Struct
DrawingObject
DrawingLine
Struct
DrawingObject
public
public
DrawingCircle
Struct
DrawingObject
DrawingRect
Struct
DrawingObject
public
public
SnapPoint
Struct
DrawingObject
DrawingTriangle
Struct
DrawingObject
BaseTool
Abstract Struct
public
public
LineTool
Struct
BaseTool
RectTool
Struct
BaseTool
public
public
CircleTool
Struct
BaseTool
TriangleTool
Struct
BaseTool

Рис 11.

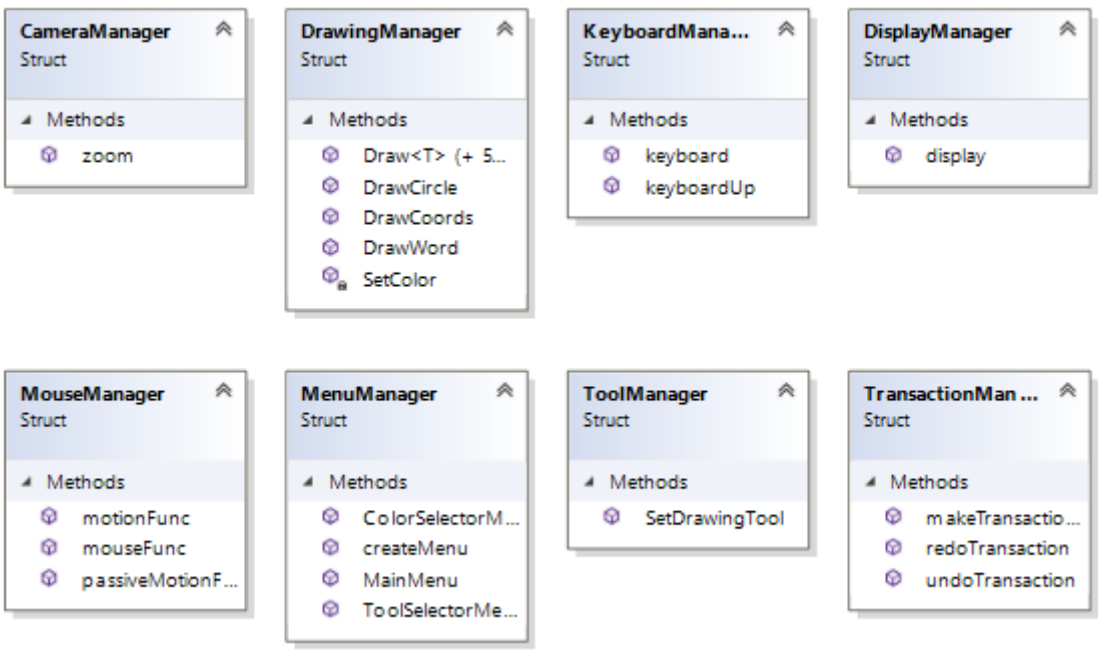

CameraManager
Struct
Methods
zoom
DrawingManager
Struct
Methods
Draw<T> (+ 5...
DrawCircle
DrawCoords
DrawWord
SetColor
KeyboardMana...
Struct
Methods
keyboard
keyboardUp
DisplayManager
Struct
Methods
display
MouseManager
Struct
Methods
motionFunc
mouseFunc
passiveMotionF...
MenuManager
Struct
Methods
ColorSelectorM...
createMenu
MainMenu
ToolSelectorMe...
ToolManager
Struct
Methods
SetDrawingTool
TransactionMan...
Struct
Methods
makeTransactio...
redoTransaction
undoTransaction

Рис 12.

4. Як розвивати далі

Система є досить гнучкою, саме тому є багато різних шляхів до її розширення. Додати нові об'єкти для малювання або інструменти для малювання буде не проблемою. Але яка ціль крім розгляду архітектури може нести в собі система. Як приклад можна розробити систему для навчання геометрії для дітей. Тобто у студента є завдання в якому потрібно побудувати графік. Студент будує графік і на основі того що він малює система підправляє його графік, тобто допомагає йому побудувати його правильно. При цьому можна піти далі і розвиватись в сторону розпізнання образів, щоб графік студента був довільним, а система сама здогадувалась яку фігуру (об'єкт) він намалював.

Можна додати можливість будувати графік за допомогою формул (по типу wolfram)

Можна також піти в сторону стереометрії, та добудувати 3D функціонал. Все що для цього потрібно, це налаштувати OpenGl та додати 3D об’єкти та інструкції як їх малювати.

5. Висновок

Отже, були досліджені різні технології для виконання системи для малювання 2D геометрії. А саме DirectX та OpenGL, та обрано OpenGL через те, що ця бібліотека досить проста у використанні (в порівнянні з DirectX), а також вона є крос-платформною. Також була обрана бібліотека для взаємодії з екраном, клавіатурою та мишкою Glut, тому що вона зручна у використанні, вона сумісна с OpenGL, а також у неї досить вичерпна документація

Була розроблена архітектура системи для малювання 2D геометрії як шаблон для подібних систем. Тобто використовуючи цю імплементацію можна зробити майже будь-яку систему для малювання геометрії.

Була розроблена система для малювання в якій можна малювати лінії, круги, трикутники, прямокутники, тощо. В якій є різні можливості для додавання видалення об’єктів, можливість повертати дії, тобто виконувати undo, redo.

6. Список використаних джерел

«GLUT Programming: Windows and Animation» - Miguel Angel Sepulveda
http://www.linuxfocus.org/Russian/January1998/article16.html

Computer Graphics with OpenGL. — 3-е изд. — М.: Вильямс, 2005.
https://works.doklad.ru/view/9pdPge_5_Xs.html

"The Mediator design pattern - Problem, Solution, and Applicability". w3sDesign.com. Retrieved 2017-08-12.
http://w3sdesign.com/?gr=b05&ugr=proble

Masterminds of Programming: Conversations with the Creators of Major Programming Languages, O'Reilly Media, Inc., 21 бер. 2009–496 стор.

https://uk.wikipedia.org/wiki/Resource_Acquisition_Is_Initialization