Графічна система для навчання

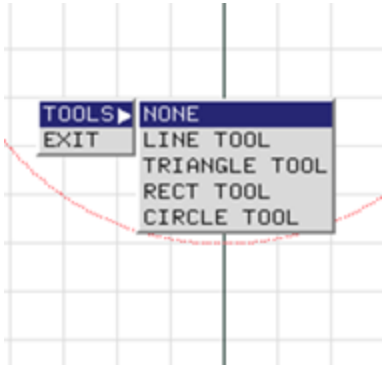
Виконав: Димченко Олексій
Керівник: Жежерун О. П.

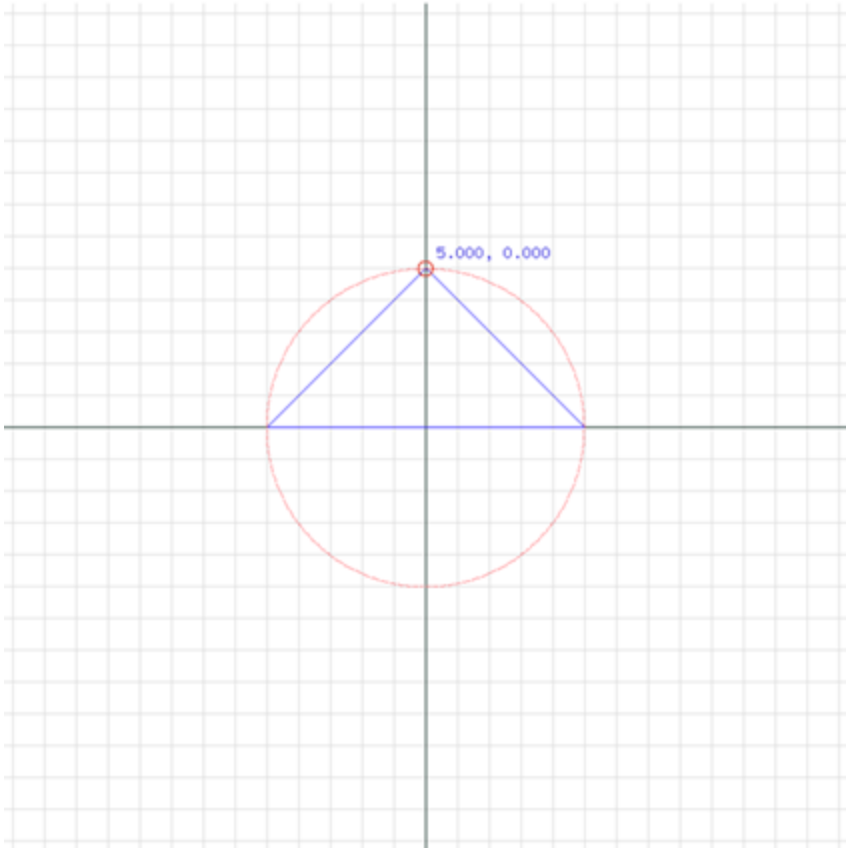
Мета

1. Порівняти та обрати технологію для відображення
2. Розробити архітектуру застосунку для малювання 2D геометрії, як шаблон для подібних застосунків
3. Розвинути додаток в навчальний

Додаток

Представляє собою інтерфейс для малювання різних геометричних примітивів





Використані технології

C++ ?

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

Використані технології

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

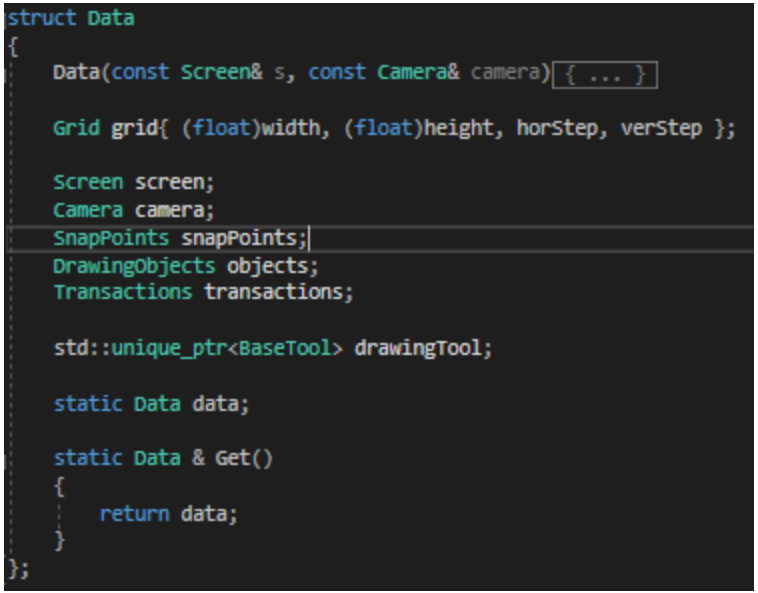
Використані технології

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. Задовольняє всі потреби для розробки системи

Архітектура

Дані знаходяться в об’єкті Data, який є Singleton



```
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;
    }
};
```

Архітектура

Об’єкти які можуть бути намальовані зберігаються в класі DrawingObjects, який є Storage

```
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);
```

```
template<class T>
std::vector<std::shared_ptr<T>>& Get();

template<>
std::vector<std::shared_ptr<DrawingLine>>& Get<DrawingLine>(){ ... }

template<>
std::vector<std::shared_ptr<DrawingTriangle>>& Get<DrawingTriangle>(){ ... }

template<>
std::vector<std::shared_ptr<DrawingRect>>& Get<DrawingRect>(){ ... }

template<>
std::vector<std::shared_ptr<DrawingCircle>>& Get<DrawingCircle>(){ ... }

template<class T>
void Add(const std::shared_ptr<T>& obj){ ... }

template<class T>
std::shared_ptr<T> Delete(const GUID g){ ... }
```

Архітектура

Менеджери це класи які відповідають за зміну даних, проте вони самі не мають станів, є Mediator

```
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) { ... }
```

Архітектура

Інструменти для малювання 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;
};
```

Архітектура

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 (RAII).

```
struct ITransaction
{
    virtual ~ITransaction() { }

    virtual void redo() = 0;
    virtual void undo() = 0;
};
```

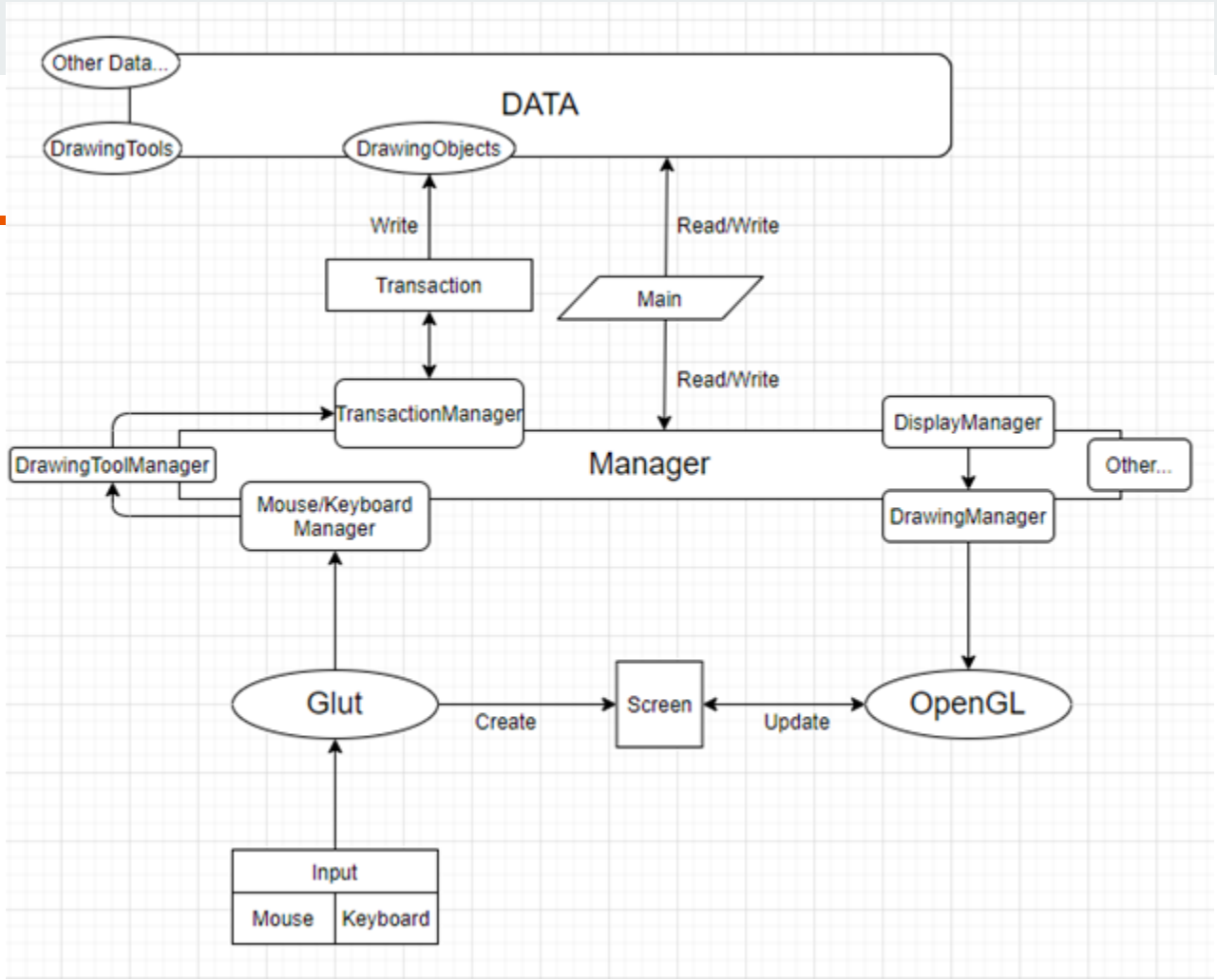
```
template<class T>
struct TransactionCreate : public ITransaction
{
    template<class... Args>
    TransactionCreate(Args&&... args)
    {
        Data::Get().objects.Add<T>(std::make_shared<T>(std::forward<Args>(args)...));
        tObj = Data::Get().objects.Get<T>().back();
    }

    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;
};
```


Other Data...
DATA
DrawingTools
DrawingObjects
Write
Read/Write
Transaction
Main
Read/Write
TransactionManager
DisplayManager
DrawingToolManager
Manager
Other...
Mouse/Keyboard Manager
DrawingManager
Glut
Create
Screen
Update
OpenGL
Input
Mouse
Keyboard

Що далі ?

1. Додати розпізнавання фігур
2. Стереометрія (3D)
3. Будувати графік за допомогою формул
4. ...

Висновок

1. Досліджені технології - DirectX та OpenGL. Обраний OpenGL
   - простий у використанні (в порівнянні з DirectX)
   - багато туторіалів
   - багато додаткових бібліотек

2. Обрана та досліджена та бібліотека Glut для взаємодії з юзером

3. Розроблена архітектура застосування для малювання 2D геометрії

4. Розроблена сама система для малювання використовуючи обрані технології

Питання ?