

**Комитет по образованию г. Санкт-Петербург**

**ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБЩЕОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ**

**ПРЕЗИДЕНТСКИЙ ФИЗИКО-МАТЕМАТИЧЕСКИЙ  
ЛИЦЕЙ №239**

**Отчет о практике  
«Создание графических приложений на язык C++»**

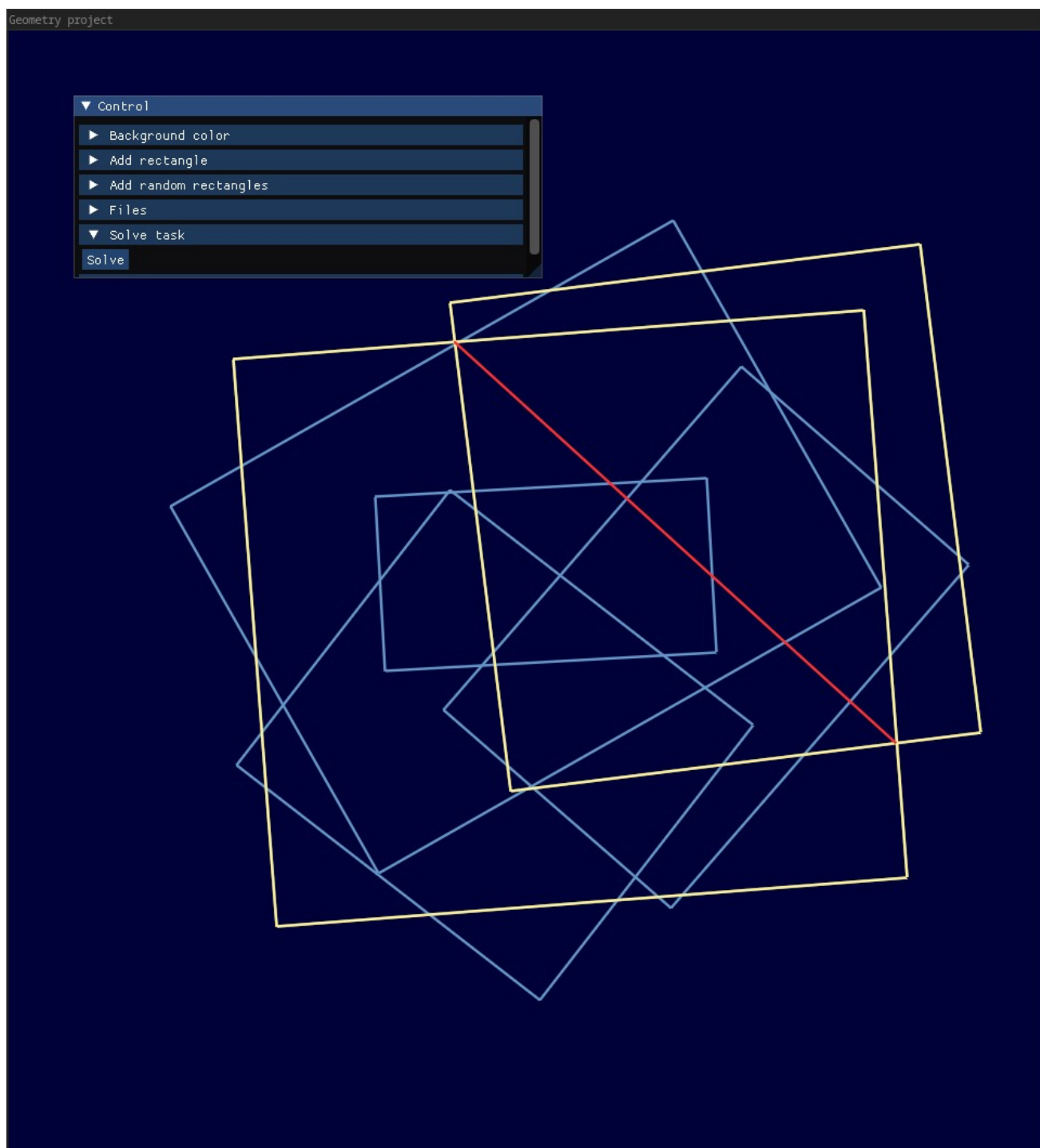
Учащийся 10-1 класса  
Сырцов Степан

Преподаватель:  
Клюнин А. О.

Санкт-Петербург – 2022 год

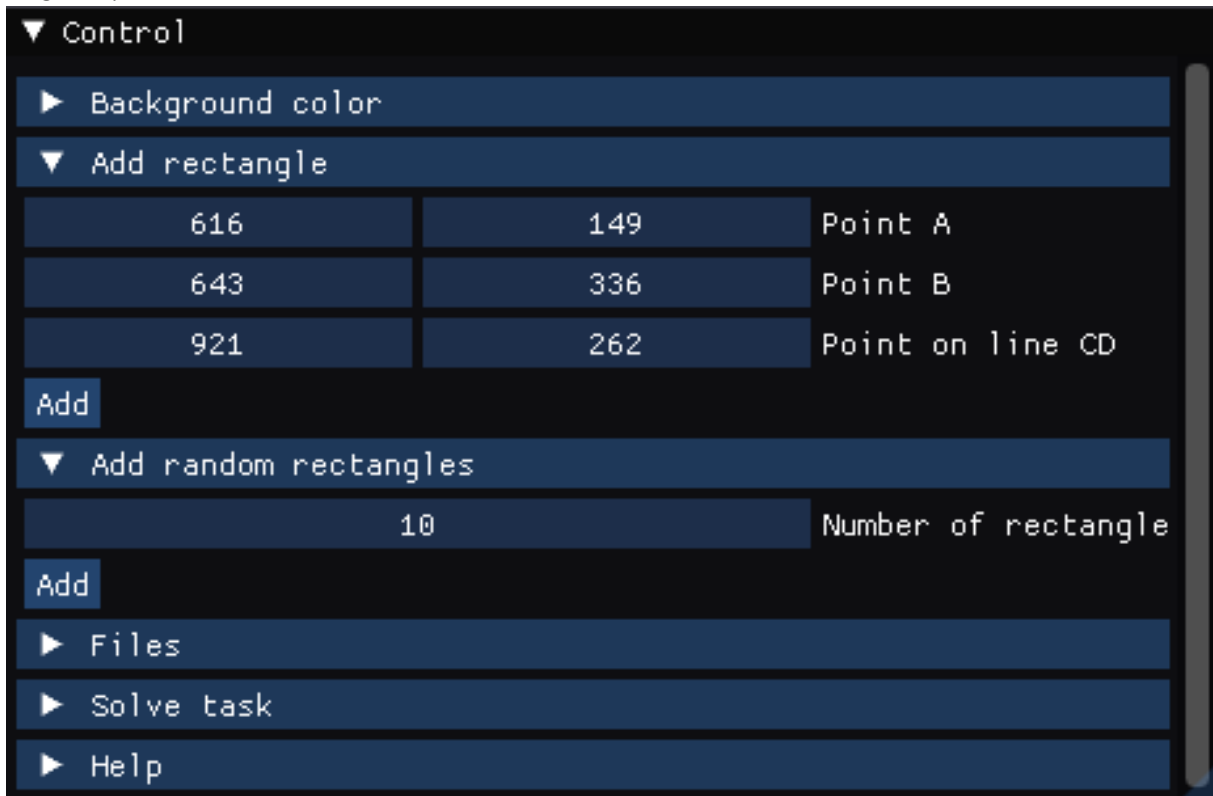
## 1. Постановка задачи

На плоскости задано множество прямоугольников. Найти такую пару пересекающихся прямоугольников, что длина отрезка, проведенного от одной точки пересечения этих двух прямоугольников до другой, максимальна. Если прямоугольники имеют более двух точек пересечения, выбирать среди них такую пару, расстояние между которыми максимально.



## 2. Элементы управления

В рамках данной задачи было необходимо реализовать следующие элементы управления:



The screenshot shows a control panel with the following elements:

- Control** (dropdown menu)
- Background color** (input field)
- Add rectangle** (dropdown menu)
- |     |     |                  |
|-----|-----|------------------|
| 616 | 149 | Point A          |
| 643 | 336 | Point B          |
| 921 | 262 | Point on line CD |
- Add** (button)
- Add random rectangles** (dropdown menu)
- |    |                     |
|----|---------------------|
| 10 | Number of rectangle |
|----|---------------------|
- Add** (button)
- Files** (input field)
- Solve task** (input field)
- Help** (input field)

Для добавления прямоугольника по координатам было создано три поля ввода: «Point A», «Point B», «Point on line CD». Так как предполагается только один вид прямоугольников, достаточно одной кнопки добавления прямоугольника и одного поля ввода для добавления случайных элементов. В него вводится количество случайных прямоугольников, которое будет добавлено. Также программа позволяет добавлять с помощью клика мышью по области рисования.

### 3. Структуры данных

Для того, чтобы хранить прямоугольники, была разработана структура **Rect**. В него были добавлены поля **pointA**, **pointB**, соответствующие двум изначально заданным вершинам, **pointP**, соответствующие точке на стороне. Были добавлены поля **pointC**, **pointD** для двух оставшихся вершин и флаг **isSolved**, показывающий, посчитаны ли **pointC** и **pointD**.

// Прямоугольник

```
struct Rect{
    //Вершины
    sf::Vector2i pointA, pointB;
    //Точка на стороне
    sf::Vector2i pointP;
    //Флаг, рассчитан ли прямоугольник
    bool isSolved=false;
    //Оставшиеся вершины
    sf::Vector2f pointC={-1, 0}, pointD={0, 0};
    Rect(const sf::Vector2i &pointA, const sf::Vector2i &pointB, const sf::Vector2i &pointP) :
        pointA(pointA), pointB(pointB), pointP(pointP){}
};
```

## 4. РИСОВАНИЕ

Для рисования прямоугольников был разработан метод **rectRender()**, который рисует четыре стороны прямоугольника

```
void rectRender(Rect* rectPtr, ImColor color, ImDrawList* drawList){
    //Если вершины прямоугольника не посчитаны
    if(!rectPtr->isSolved){
        solveRect(rectPtr);
    }
    sf::Vector2i pointD=static_cast<sf::Vector2i>(rectPtr->pointD);
    sf::Vector2i pointC=static_cast<sf::Vector2i>(rectPtr->pointC);
    //Рисуем стороны прямоугольника
    drawList->AddLine(
        rectPtr->pointA, rectPtr->pointB,
        color,
        2.5
    );
    drawList->AddLine(
        rectPtr->pointB, pointC,
        color,
        2.5
    );
    drawList->AddLine(
        pointC, pointD,
        color,
        2.5
    );
    drawList->AddLine(
        pointD, rectPtr->pointA,
        color,
        2.5
    );
}
```

## 5. Решение задачи

Для решения задачи был разработан метод **solveTask()**.

//Решение задачи

```
void solveTask(){
    //По всем прямоугольникам
    for(int i=maxSolvedRect+1; i<rectangles.size(); i++){
        //Считаем прямоугольник, если нужно
        if(!rectangles.at(i).isSolved){
            solveRect(&rectangles.at(i));
        }
        Rect * pRect1=&rectangles.at(i);
        //Создаём массив с вершинами
        sf::Vector2f rect1Points[4]={static_cast<sf::Vector2f>(pRect1->pointA),
        static_cast<sf::Vector2f>(pRect1->pointB), pRect1->pointC, pRect1->pointD};
        //По всем прямоугольникам
        for(int j=0; j<i; j++){
            Rect * pRect2=&rectangles.at(j);
            //Создаём массив с вершинами
            sf::Vector2f rect2Points[4]={static_cast<sf::Vector2f>(pRect2->pointA),
            static_cast<sf::Vector2f>(pRect2->pointB), pRect2->pointC, pRect2->pointD};
            std::vector<sf::Vector2f> possiblePoints;
            //Пересекаем все стороны прямоугольников со всеми
            for(int ii=0; ii<4; ii++){
                for(int jj=0; jj<4; jj++){
                    intersectSegments(rect1Points[ii], rect1Points[(ii+1)%4],
                    rect2Points[jj], rect2Points[(jj+1)%4], &possiblePoints);
                }
            }
            //Сравниваем расстояния между точками пересечения
            for(auto point1:possiblePoints){
                for(auto point2:possiblePoints){
                    if(abs(point1-point2)>maxDistance){
                        maxDistance=abs(point1-point2);
                        maxPoint1=point1;
                        maxPoint2=point2;
                        rect1Num=i;
                        rect2Num=j;
                    }
                }
            }
        }
    }
    //Записываем количество сравненных между собой прямоугольников
    maxSolvedRect=rectangles.size()-1;
}
```

## 6. Заключение

В рамках выполнения поставленной задачи было создано графическое приложение с требуемым функционалом.