概要设计

开发环境

- IDE：Visual Studio 2019 Commity
- 运行环境：window10 专业版
- 配置要求：内存 4g

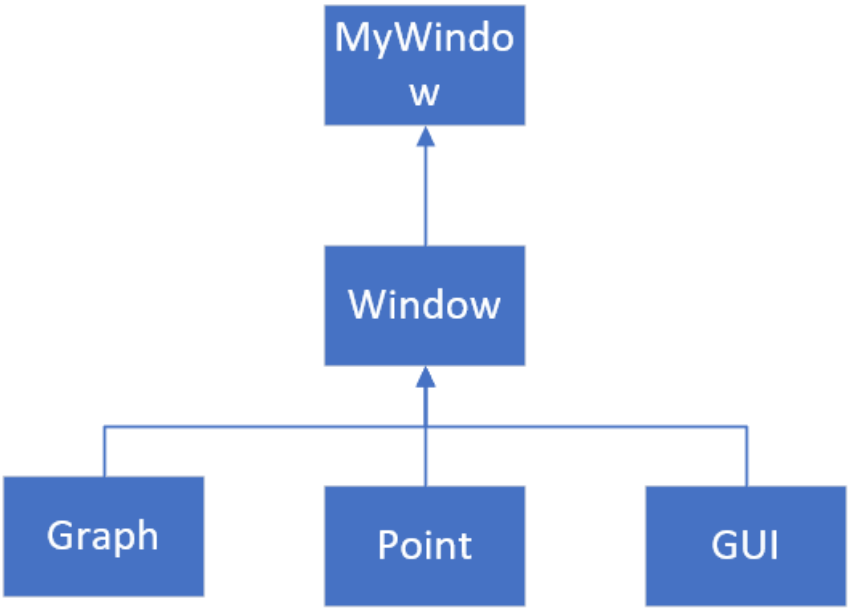
  显卡 无要求

  CPU 无要求

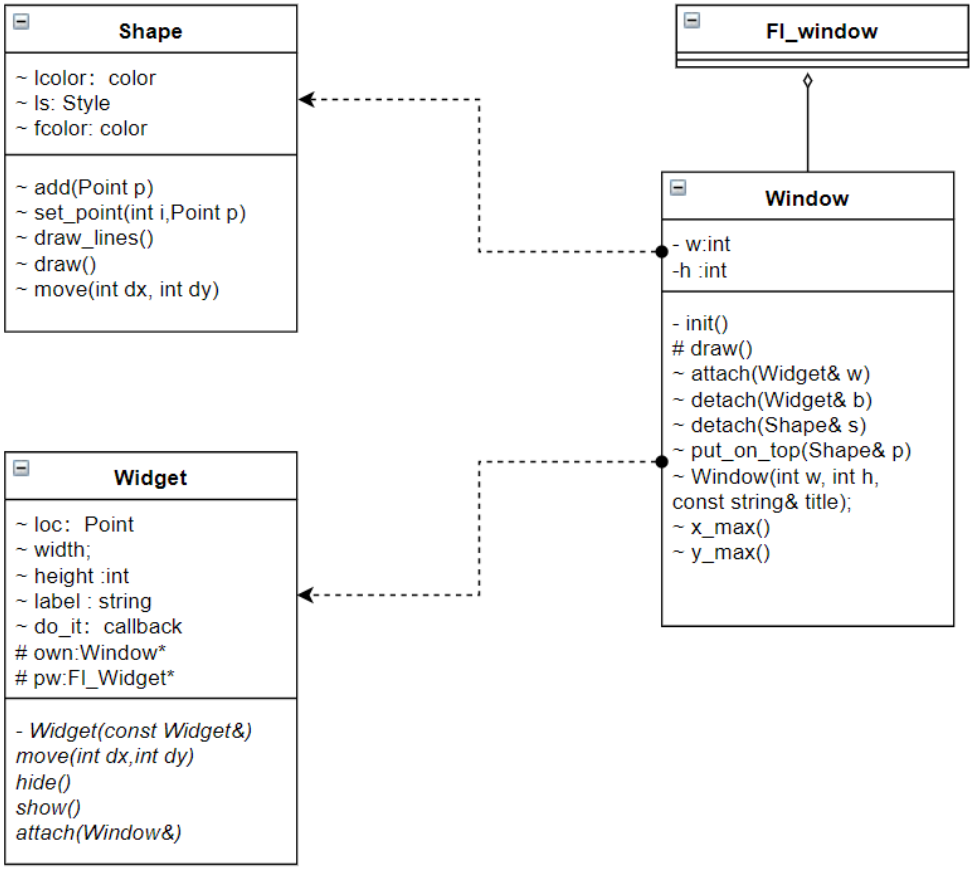
目的

调试编译通过图形应用编码、熟悉flk

结构化模块设计图



UML类图



主要模块功能接口描述

Graph.cpp

```
void Shape::add(Point p) // 向图形中添加点
void Shape::draw_lines() // 绘制线条
void Shape::draw() // 根据选定的线条样式、颜色等绘制图像
void Shape::move(int dx, int dy) // 图像移动 dx和dy是图像横坐标和纵坐标移动的截距
void Lines::add(Point p1, Point p2) // 添加线
void Lines::draw_lines() const // 根据里面的点绘制线条
inline pair<double,double> line_intersect(Point p1, Point p2, Point p3, Point
p4, bool& parallel) // 线条之间的相交
void Polygon::add(Point p) // 多边形的添加点
void Polygon::draw_lines() // 画线
void draw_mark(Point xy, char c) // 绘制标记
void Rectangle::draw_lines() const // 矩形线条绘制
Point Circle::center() // 圆心
void Circle::draw_lines() // 绘制圆的线条
void Ellipse::draw_lines() const // 椭圆画线
void Axis::draw_lines() const // 坐标轴画线
void Axis::set_color(Color c) // 设置颜色
void Axis::move(int dx, int dy) //axis移动
```

GUI.cpp

```
void Button::attach(Window& win) // 按钮连接到窗口上
int Menu::attach(Button& b) // menu连接到窗口上
```

Simple_window.cpp

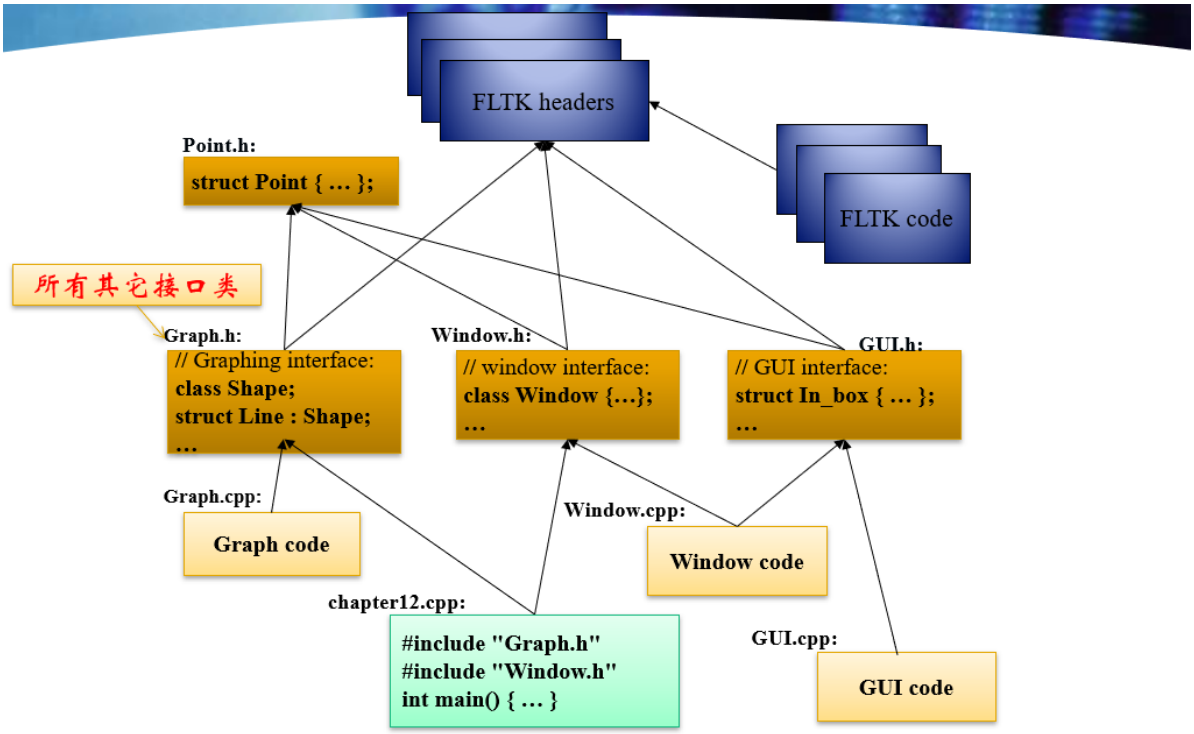
```
bool Simple_window::wait_for_button() // 等到按钮的点击事件
void Simple_window::cb_next(Address, Address pw) // 将Simple_window::next()连接到
pw
void Simple_window::next()
```

window.cpp

```
void Window::draw() // 窗口绘制
void Window::attach(Widget& w) // 窗口链接
void Window::detach(Widget& b) // 取消链接
```

详细设计与实现

代码组织图



核心数据结构描述

Point.h

```
struct Point { // 创建一个点结构体 用来存放点的坐标信息 以左上角为（0，0）Point默认（0,0）
    int x, y;
    Point(int xx, int yy) : x(xx), y(yy) { }
    Point() :x(0), y(0) { }
};
```

Simple_window.h

```
struct Simple_window : Window {  // 简单窗口结构体
    Simple_window(Point xy, int w, int h, const string& title ); // 包含初始点信息，w表示宽度，h表示高度，title表示窗口标题

    bool wait_for_button(); // 等待按钮点击事件

private:
    Button next_button;     // next按钮
    bool button_pushed;     // implementation detail

    static void cb_next(Address, Address); // next_button的回调函数
    void next();            // next点击后需要执行的事件
};
```

Graph.h

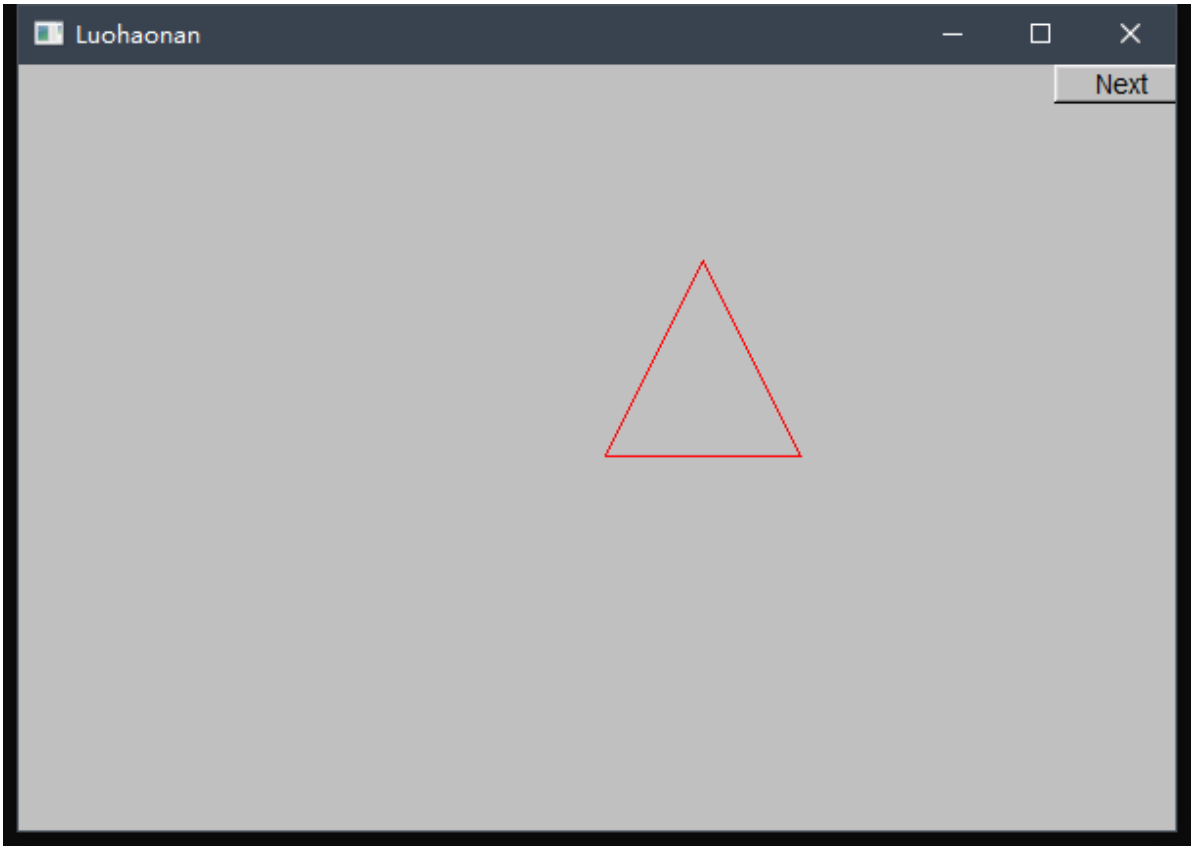
```
struct Color { // 颜色的结构体
    enum Color_type { //包含各种颜色的枚举类型
        red=FL_RED,
        blue=FL_BLUE,
        green=FL_GREEN,
        yellow=FL_YELLOW,
        white=FL_WHITE,
        black=FL_BLACK,
        magenta=FL_MAGENTA,
        cyan=FL_CYAN,
        dark_red=FL_DARK_RED,
        dark_green=FL_DARK_GREEN,
        dark_yellow=FL_DARK_YELLOW,
        dark_blue=FL_DARK_BLUE,
        dark_magenta=FL_DARK_MAGENTA,
        dark_cyan=FL_DARK_CYAN
    };

    enum Transparency { invisible = 0, visible=255 };

    Color(Color_type cc) :c(Fl_Color(cc)), v(visible) { }
    Color(Color_type cc, Transparency vv) :c(Fl_Color(cc)), v(vv) { }
    Color(int cc) :c(Fl_Color(cc)), v(visible) { }
    Color(Transparency vv) :c(Fl_Color()), v(vv) { }    // default color

    int as_int() const { return c; }

    char visibility() const { return v; }
    void set_visibility(Transparency vv) { v=vv; }
private:
    char v;    // invisible and visible for now
    Fl_Color c;
};

struct Line_style {  // 线的样式的结构体
    enum Line_style_type { // 线的样式的枚举
        solid=FL_SOLID,           // -------
        dash=FL_DASH,             // - - - -
        dot=FL_DOT,               // .......
        dashdot=FL_DASHDOT,       // - . - .
```

```
        dashdotdot=FL_DASHDOTDOT,  // -..-..
    };

    Line_style(Line_style_type ss) :s(ss), w(0) { }
    Line_style(Line_style_type lst, int ww) :s(lst), w(ww) { }
    Line_style(int ss) :s(ss), w(0) { }

    int width() const { return w; }
    int style() const { return s; }
private:
    int s;
    int w;
};

struct Rectangle : Shape { // 矩形的结构体
    Rectangle(Point xy, int ww, int hh) : w(ww), h(hh) // 包含锚点 宽度和高度
    {
        add(xy);
        if (h<=0 || w<=0) error("Bad rectangle: non-positive side");
    }

    Rectangle(Point x, Point y) : w(y.x-x.x), h(y.y-x.y)
    {
        add(x);
        if (h<=0 || w<=0) error("Bad rectangle: non-positive width or height");
    }
    void draw_lines() const;

    int height() const { return h; }
    int width() const { return w; }
private:
    int h;    // height
    int w;    // width
};

struct Open_polyline : Shape {         // open sequence of lines
    void add(Point p) { Shape::add(p); }
    void draw_lines() const;
};

//------------------------------------------------------------------------------

struct Text : Shape { // 文本
    // 锚点在文字的第一个字的左下方
    Text(Point x, const string& s) : lab(s), fnt(fl_font()), fnt_sz(fl_size()) {
add(x); }

    void draw_lines() const;

    void set_label(const string& s) { lab = s; }
    string label() const { return lab; }

    void set_font(Font f) { fnt = f; }
    Font font() const { return Font(fnt); }

    void set_font_size(int s) { fnt_sz = s; }
    int font_size() const { return fnt_sz; }
private:
```

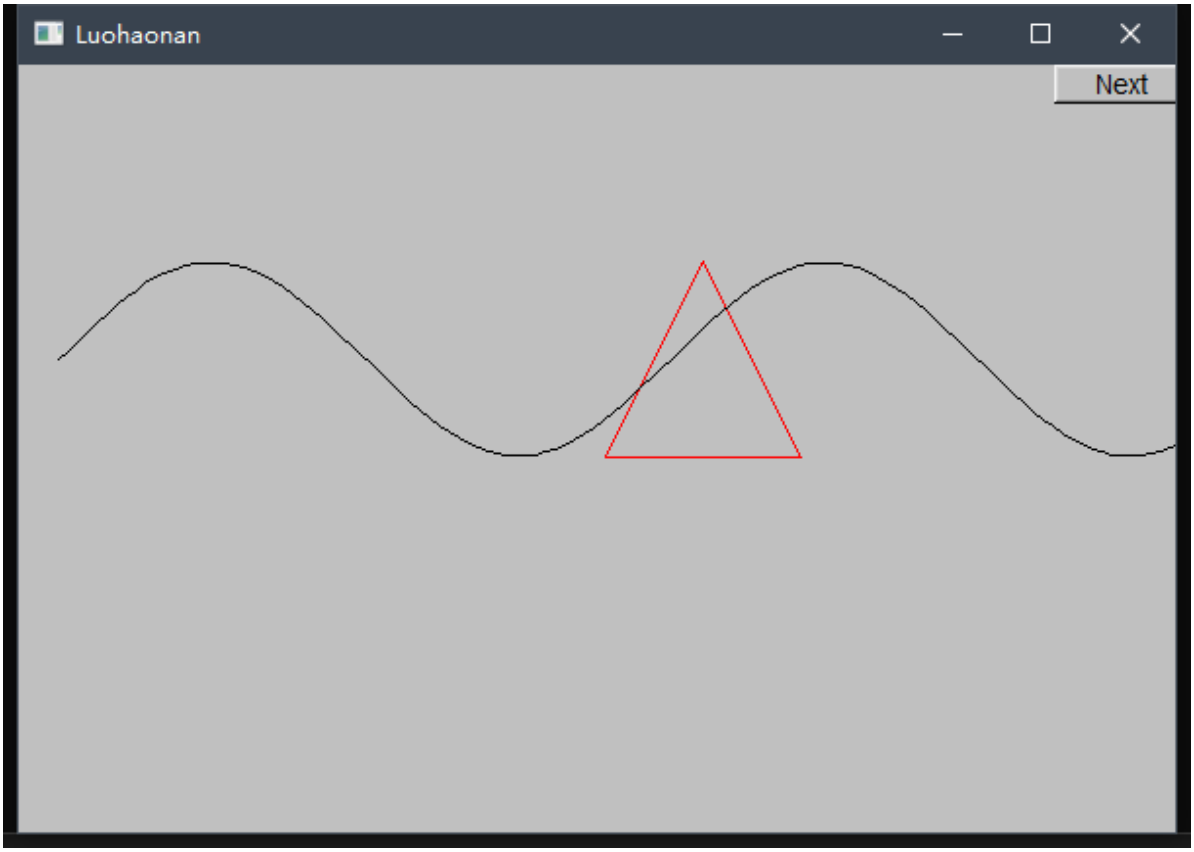
```
    string lab;    // label
    Font fnt;
    int fnt_sz;
};

//------------------------------------------------------------------------------

struct Axis : Shape { // 坐标
    enum Orientation { x, y, z };
    Axis(Orientation d, Point xy, int length,
        int number_of_notches=0, string label = "");

    void draw_lines() const;
    void move(int dx, int dy);
    void set_color(Color c);

    Text label;
    Lines notches;
};

//------------------------------------------------------------------------------

struct Circle : Shape { // 圆
    Circle(Point p, int rr);    // center and radius
    ...
};

//------------------------------------------------------------------------------

struct Ellipse : Shape { // 椭圆
    Ellipse(Point p, int w, int h)    // center, min, and max distance from
center
        : w(w), h(h)
    {
        add(Point(p.x-w,p.y-h));
    }

    ...
};

//------------------------------------------------------------------------------

struct Marks : Marked_polyline { // 标记
    ...
};

//------------------------------------------------------------------------------

struct Mark : Marks { // 标记
    Mark(Point xy, char c) : Marks(string(1,c))
    {
        add(xy);
    }
};

//------------------------------------------------------------------------------

struct Suffix { // 后缀结构体 none jpg gif
```

```
    enum Encoding { none, jpg, gif  };
};

Suffix::Encoding get_encoding(const string& s);
```

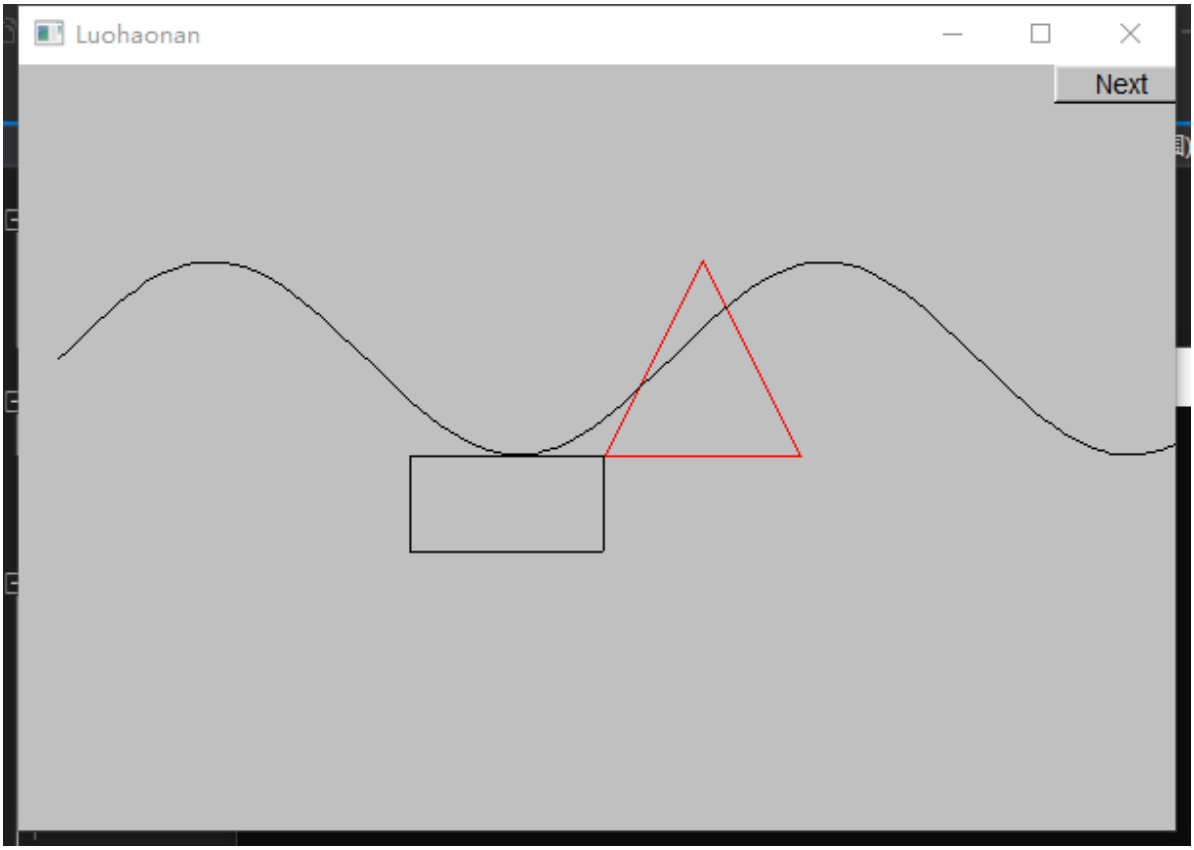
绘制图形



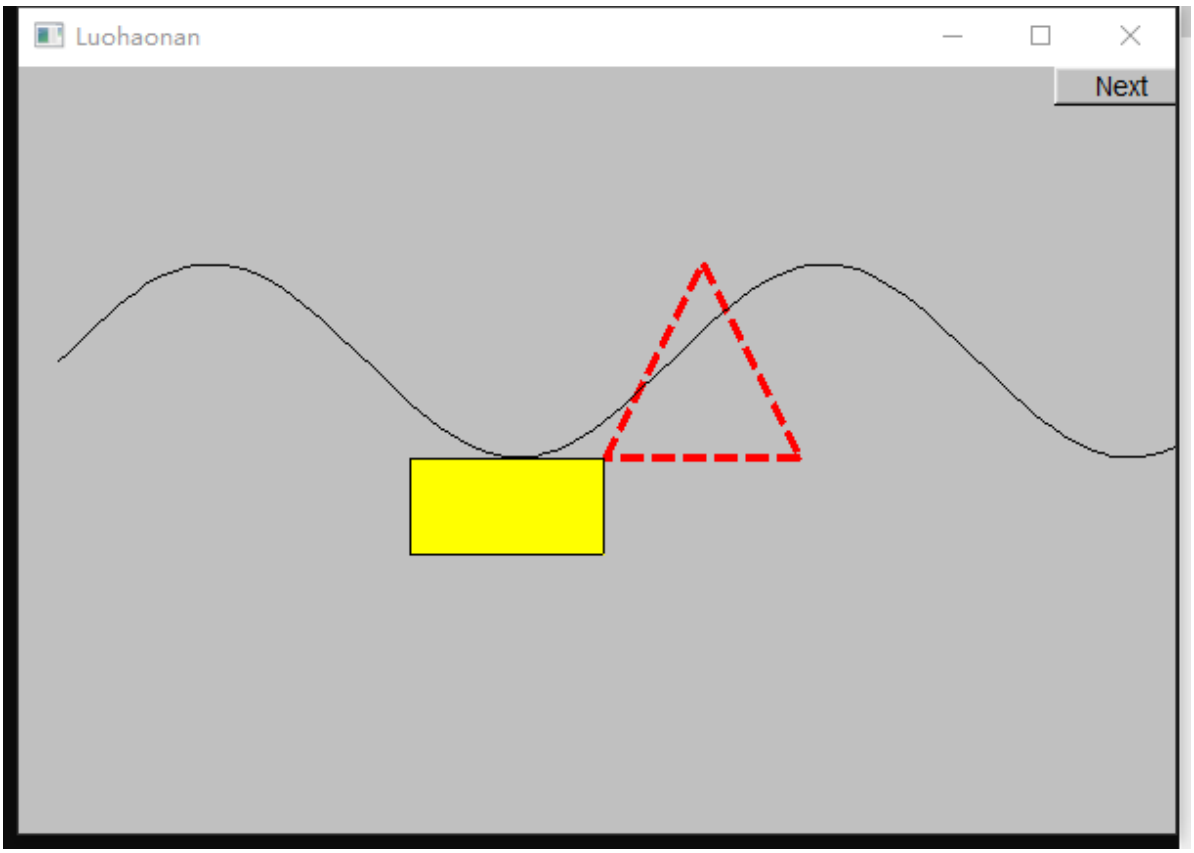
将sin添加进去



将矩形添加进去



添加颜色等



最后代码：

```
using namespace Graph_lib;   // our graphics facilities are in Graph_lib

    Point tl(100,100);           // to become top left  corner of window

    Simple_window win(tl,600,400,"Luohaonan");    // make a simple window
```

```
    Graph_lib::Polygon poly;                // make a shape (a polygon)

    poly.add(Point(300,200));    // add a point
    poly.add(Point(350,100));    // add another point
    poly.add(Point(400,200));    // add a third point

    poly.set_color(Color::red);  // adjust properties of poly

    win.attach (poly);           // connect poly to the window

    Function sine(sin, 0, 100, Point(20, 150), 1000, 50, 50);
    win.attach(sine);

    Graph_lib::Rectangle r(Point(200, 200), 100, 50);
    win.attach(r);

    Closed_polyline poly_rect;
    poly_rect.add(Point(100, 50));
    poly_rect.add(Point(200, 50));
    poly_rect.add(Point(200, 100));
    poly_rect.add(Point(100, 100));
    poly_rect.add(Point(50, 75));

    r.set_fill_color(Color::yellow);    // color the inside of the rectangle

    poly.set_style(Line_style(Line_style::dash, 4));      // make the triangle
fat

    poly_rect.set_fill_color(Color::green);
    poly_rect.set_style(Line_style(Line_style::dash, 2));

    Text t(Point(100, 100), "Hello, graphical world!");  // add text

    win.wait_for_button();       // give control to the display engine
```