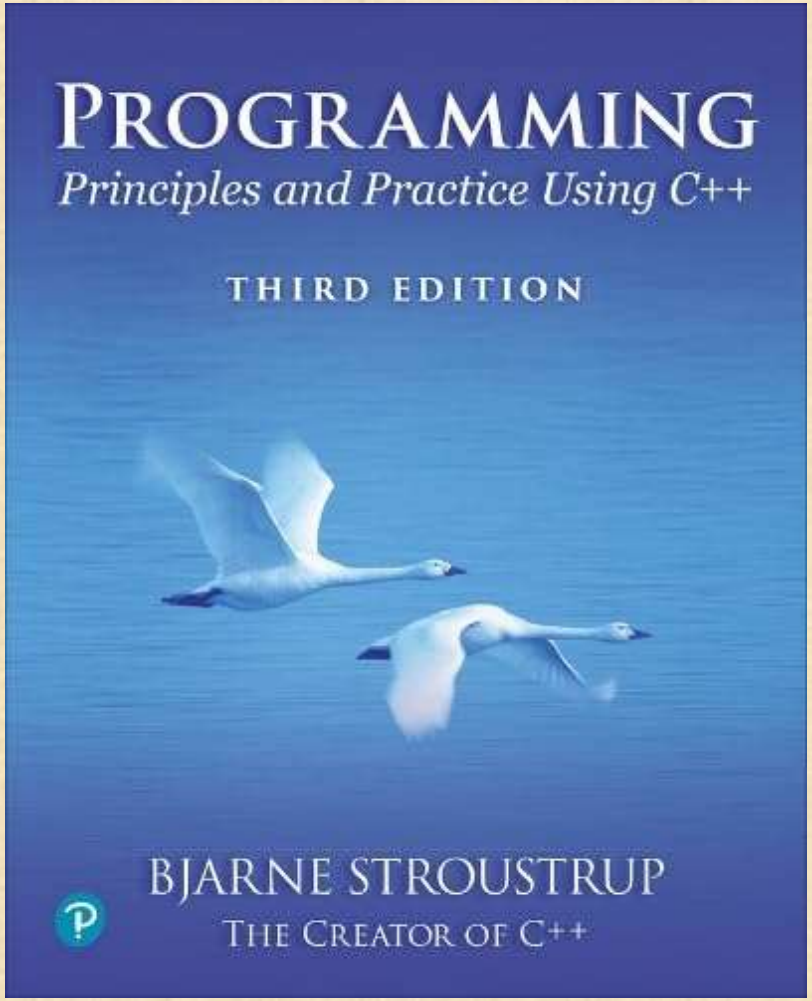
Chapter 12 – A display model



The world was black and white then.
It didn´t turn color
until sometime in the 1930s.
– Calvin´s dad

Overview

- Why graphics?
- A graphics model
- Examples

Why bother with graphics and GUI?

- It's very common
  - If you write conventional PC applications, you'll have to do it
- It's useful
  - Instant feedback
  - Graphing functions
  - Displaying results
- It can illustrate some generally useful concepts and techniques

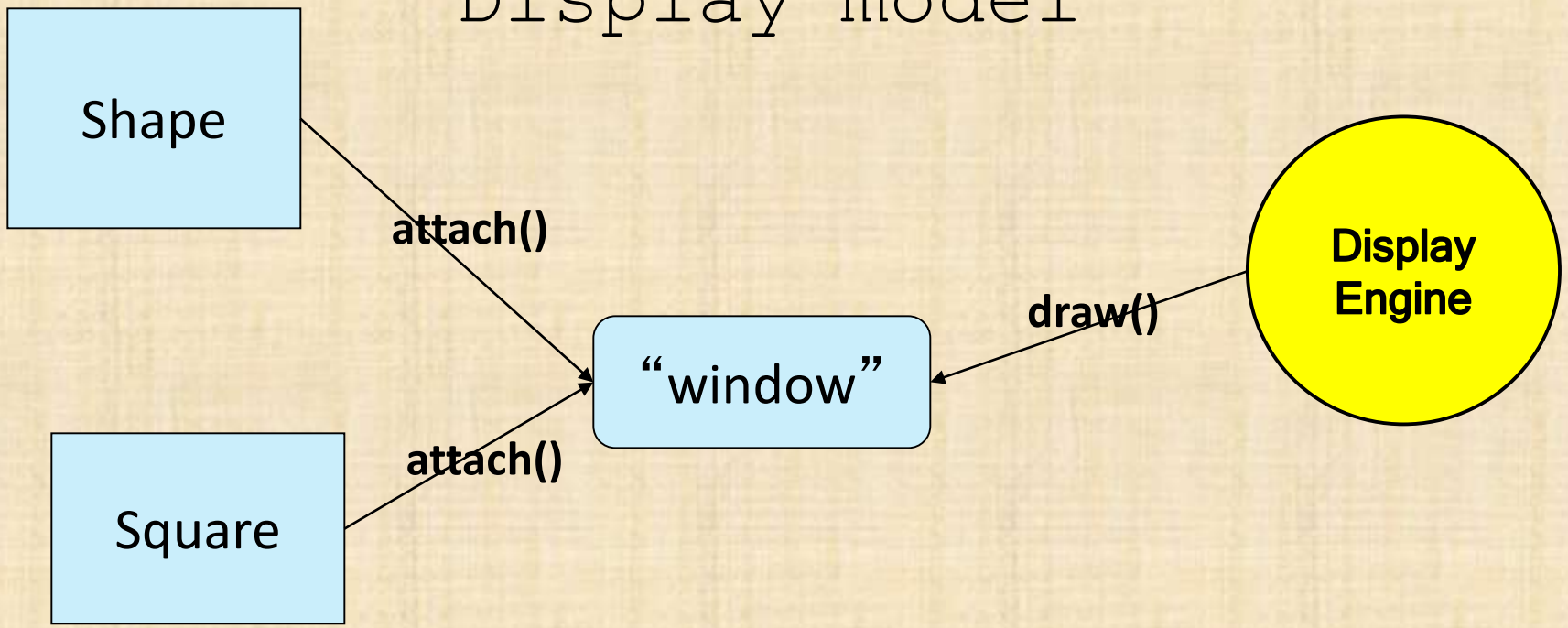
Why bother with graphics and GUI?

- It can only be done well using some pretty neat language features ☺
- Lots of good (small) code examples
- It can be non-trivial to “get” the key concepts
  - So it’s worth teaching
  - If we don’t show how it’s done, you might think it was “magic”
- Graphics is fun!

Why Graphics/GUI?

- WYSIWYG
  - What you see (in your code) is what you get (on your screen)
- Direct correspondence between concepts, code, and output

Display model



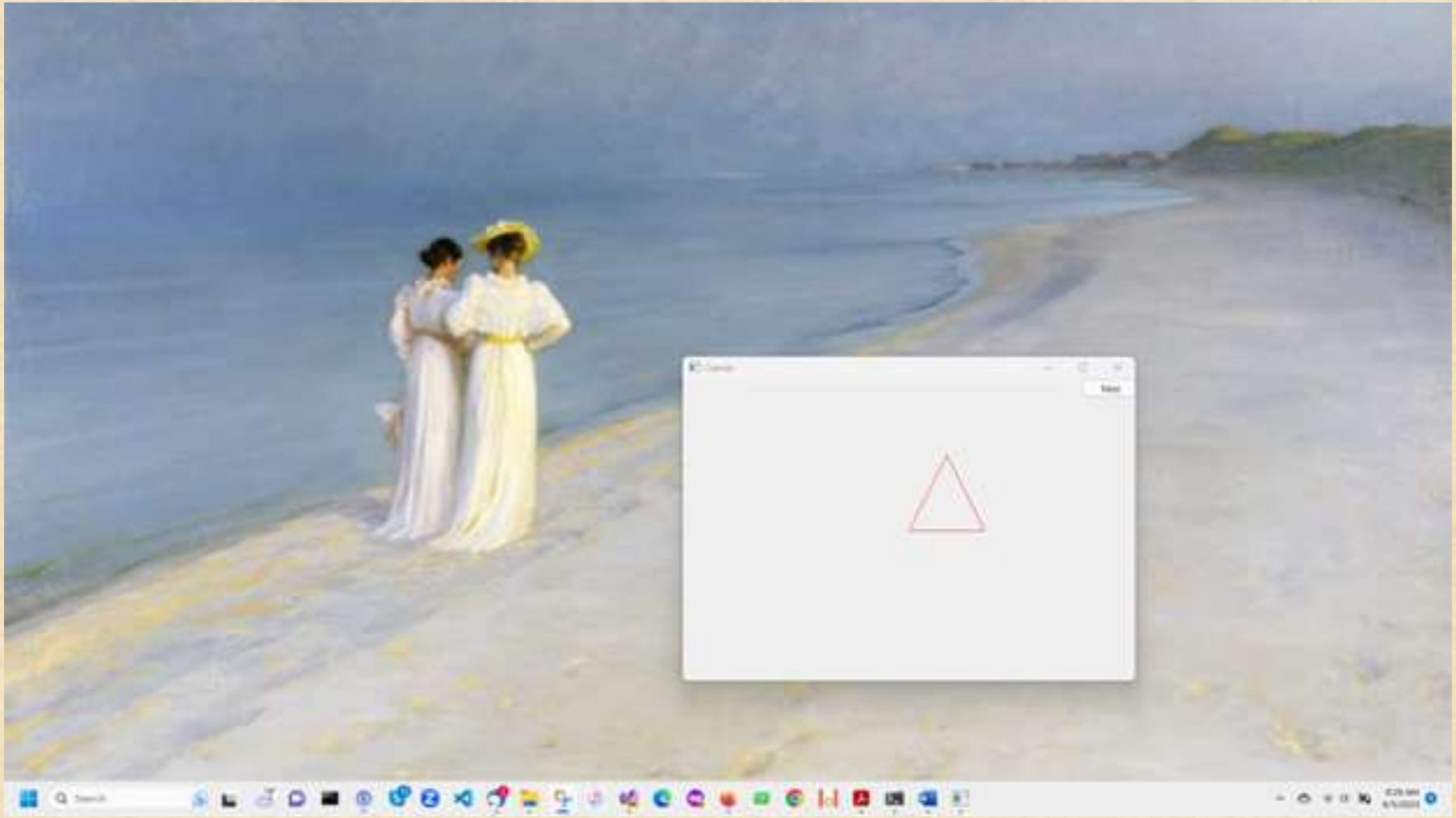
- Objects (such as graphs) are “attached to” a window.
- The “display engine” invokes display commands (such as “draw line from x to y”) for the objects in a window
- Objects such as Square contain vectors of lines, text, etc. for the window to draw

Display model

- An example illustrating the display model

```
int main()
{
  using namespace Graph_lib;          // use our graphics interface library

  Application app;                    // start a Graphics/GUI application

  Point tl {900,500};                 // to become the top left corner of the window

  Simple_window win {tl,600,400,"Canvas"};          // make a simple window

  Polygon poly;                       // make a shape (a polygon, obviously)
  poly.add(Point{300,200});           // add three points to the polygon
  poly.add(Point{350,100});
  poly.add(Point{400,200});
  poly.set_color(Color::red);         // make the polygon red (obviously)

  win.attach(poly);                   // connect poly to the window
  win.wait_for_button();              // give control to the display engine
}
```

The resulting screen



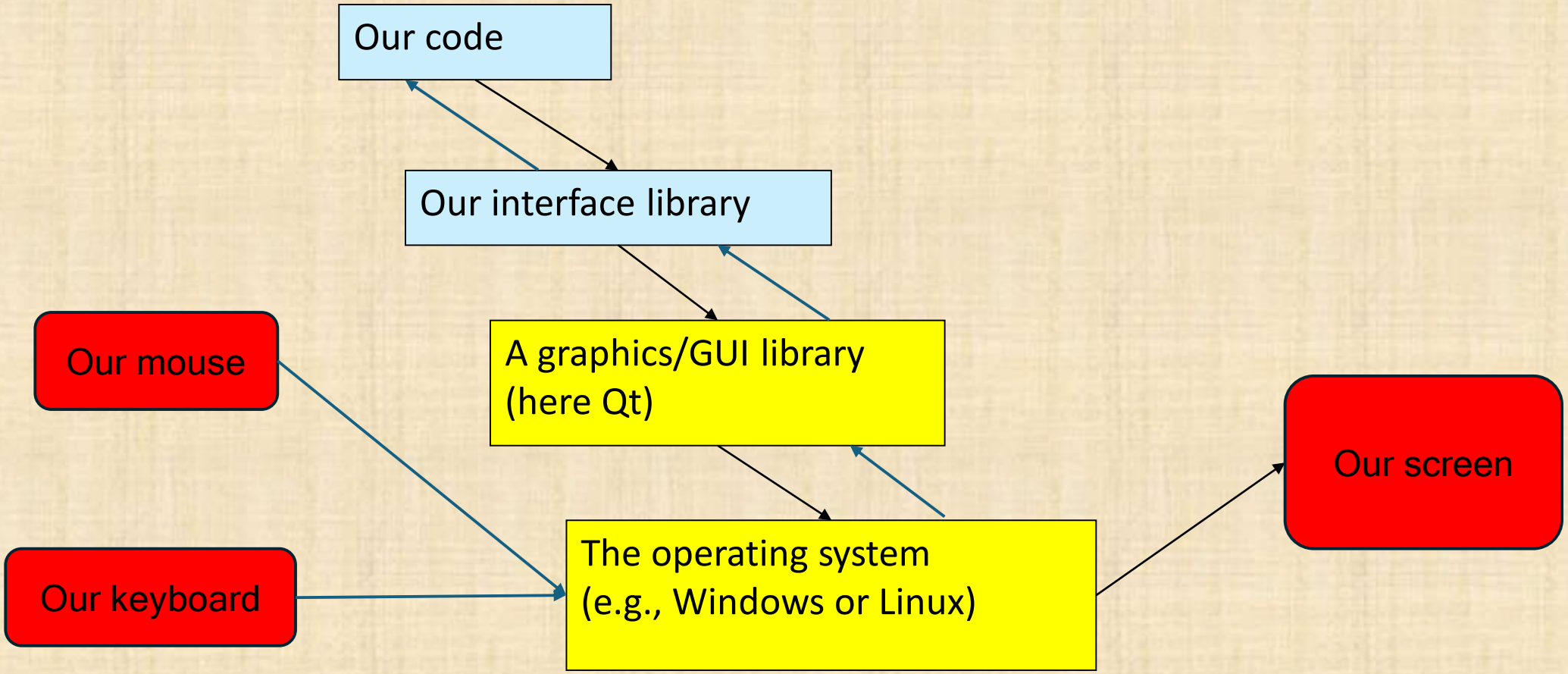
Graphics/GUI libraries

- You'll be using a few interface classes we wrote
  - Interfacing to a popular GUI toolkit
    - GUI == Graphical User Interface
    - Qt: https://en.wikipedia.org/wiki/Qt_(software)
  - Installation, etc.
    - https://github.com/villevoutilainen/ProgrammingPrinciplesAndPracticeUsingQt
    - If you can, ask an instructor or friend for assistance to install
- Our Graphics/GUI model is far simpler than common toolkit interfaces
  - Our interface library is ~20 classes and ~500 lines of code
    - Manageable for teaching / early learning
  - You can write a lot of code with these classes
    - And you can build more classes on them

Graphics/GUI libraries (cont.)

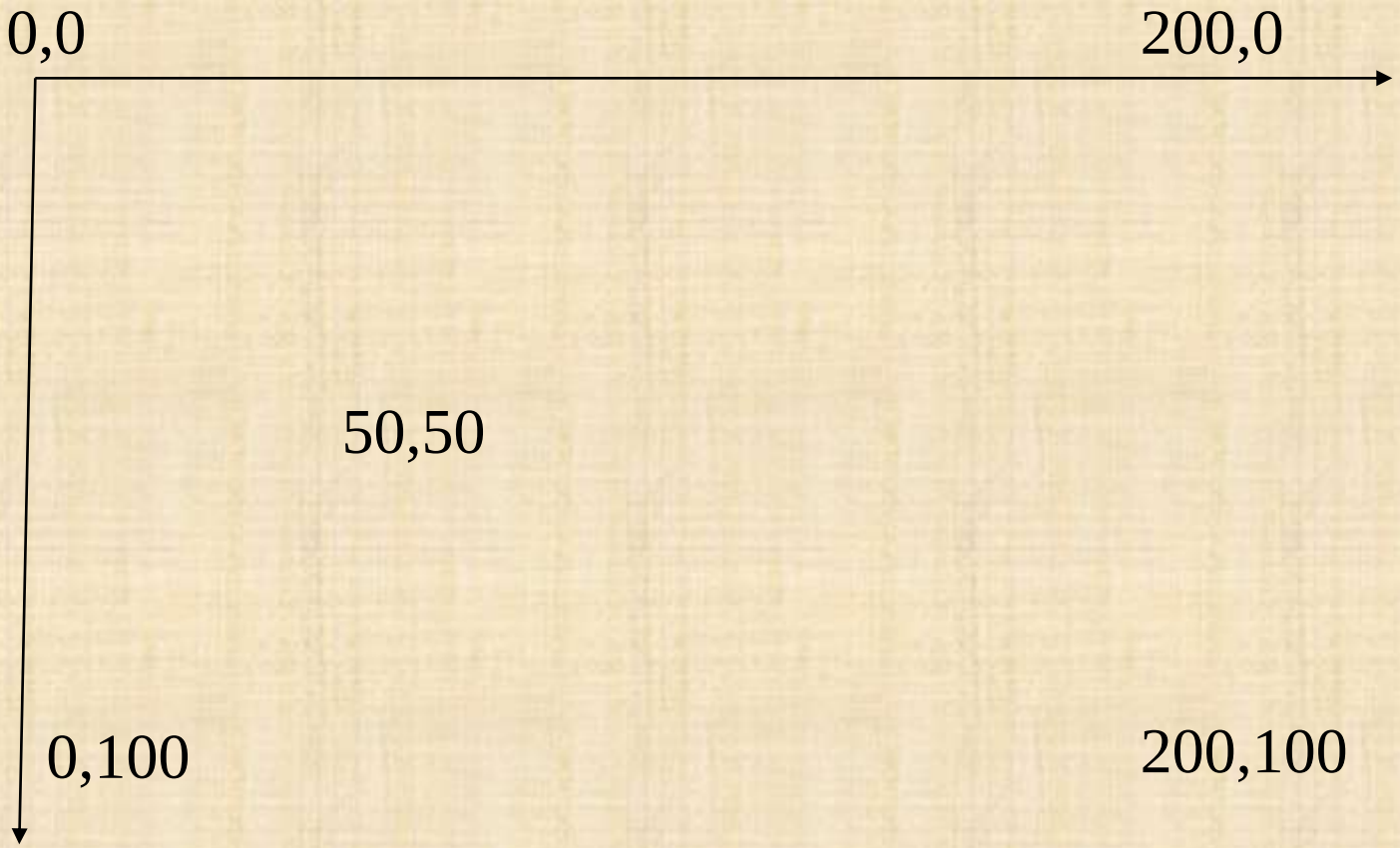
- The code is portable
  - Windows, Unix, Mac, phones, browsers, etc.
- This model extends to most common graphics and GUI uses
- The general ideas can be used with any popular GUI toolkit
  - Once you understand the graphics classes you can easily learn any GUI/graphics library
    - Well, relatively easily – these libraries are huge

Graphics/GUI libraries



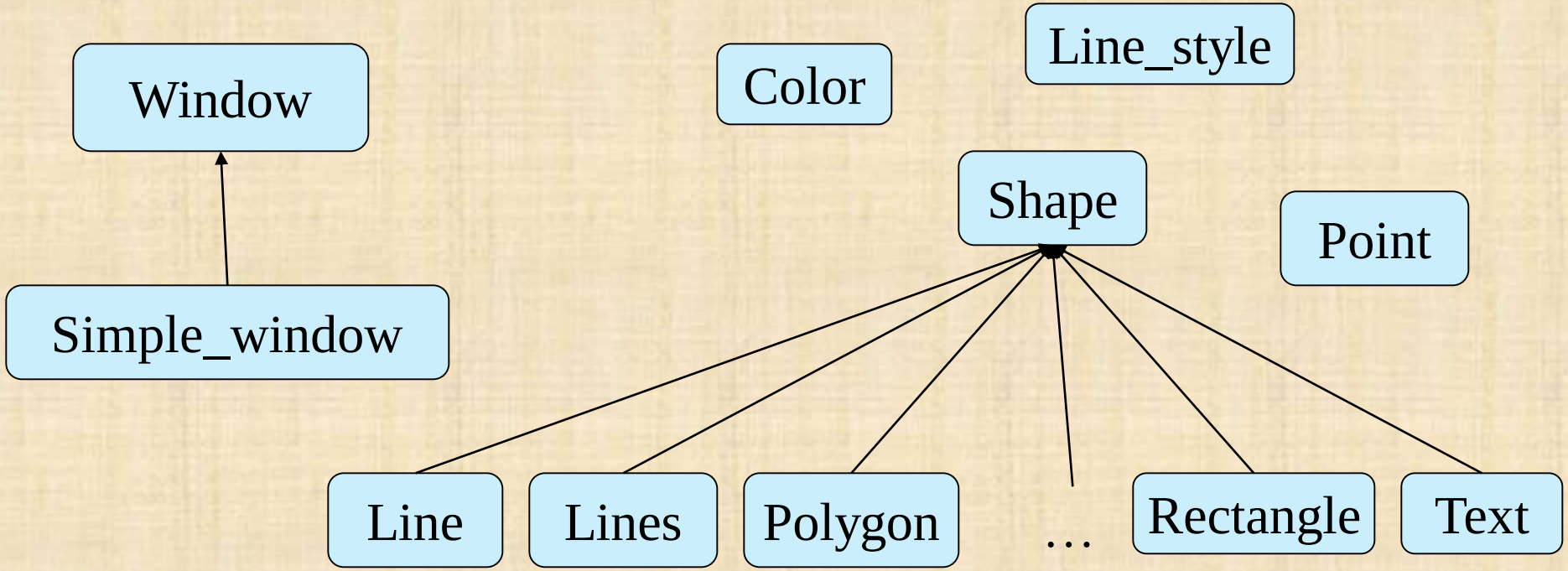
- Often called “a layered architecture”

Coordinates



- Oddly, y-coordinates "grow downwards" // *right, down*
- Coordinates identify pixels in the window on the screen
- You can resize a window (changing **x_max()** and y_**max())**

Interface classes



- An arrow means "is a kind of"
- **Color**, **Line_style**, and **Point** are "utility classes" used by the other classes
- **Window** is our interface to the GUI library (which is our interface to the screen)

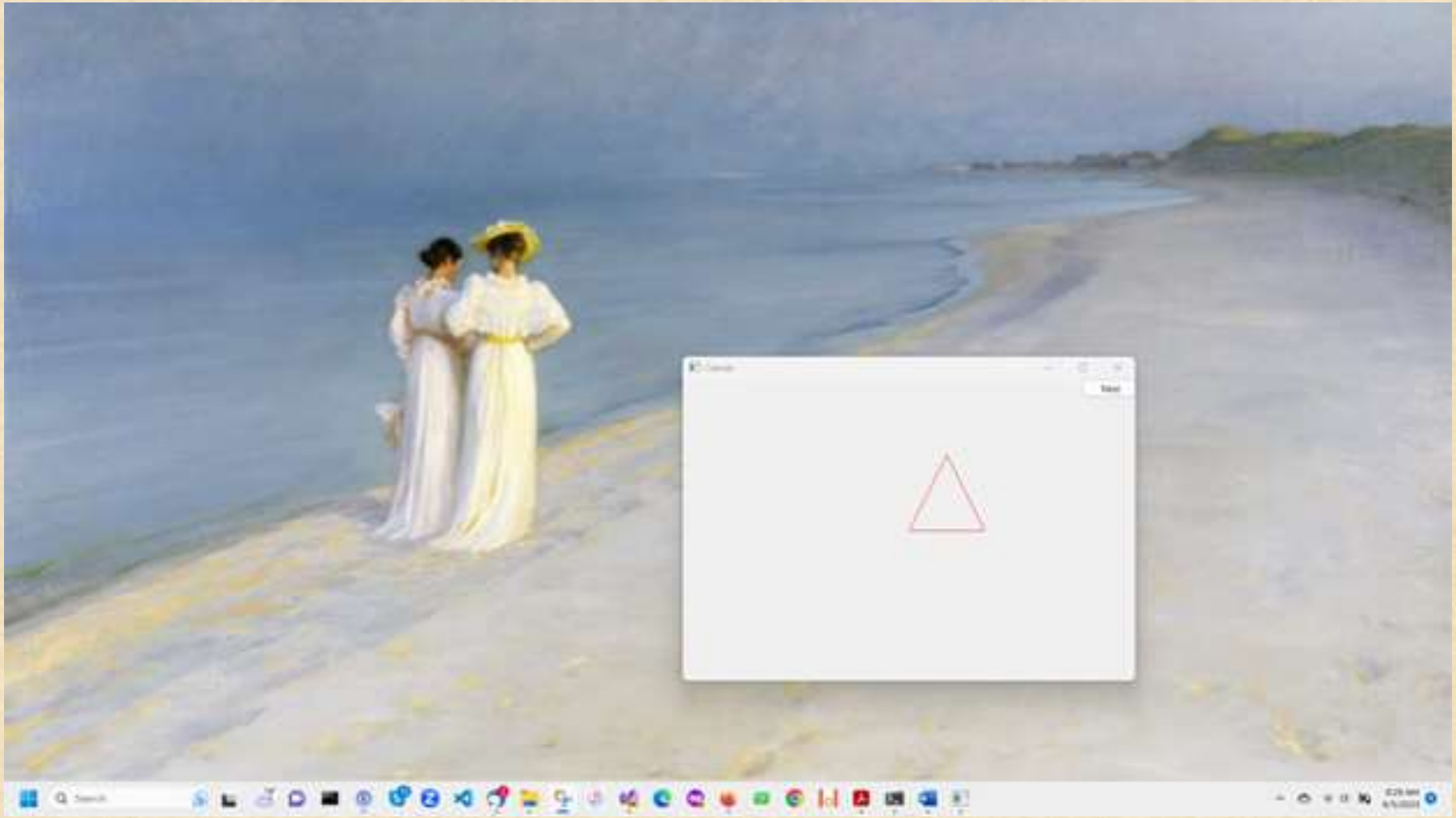
Interface classes

- Current
  - Color, Line_style, Font, Point,
  - Window, Simple_window
  - Shape, Text, Polygon, Line, Lines, Rectangle, Function, Circle, Ellipse, …
  - Axis
- Easy to add (for some definition of "easy")
  - Grid, Block_chart, Pie_chart, etc.
- Later, GUI
  - Button, In_box, Out_box, …

"Boilerplate"

```
// Getting access to the graphics system (don't forget to
 install):
#include "Simple_window.h"    // stuff to deal with your
 system's windows
#include "Graph.h"            // graphical shapes

int main()
{
 using namespace Graph_lib;   // use our graphics interface
 library

 Application app;             // start a Graphics/GUI
 application
 Simple_window win {Point{900,500},600,400,"Canvas"};  // make
 a simple window
```

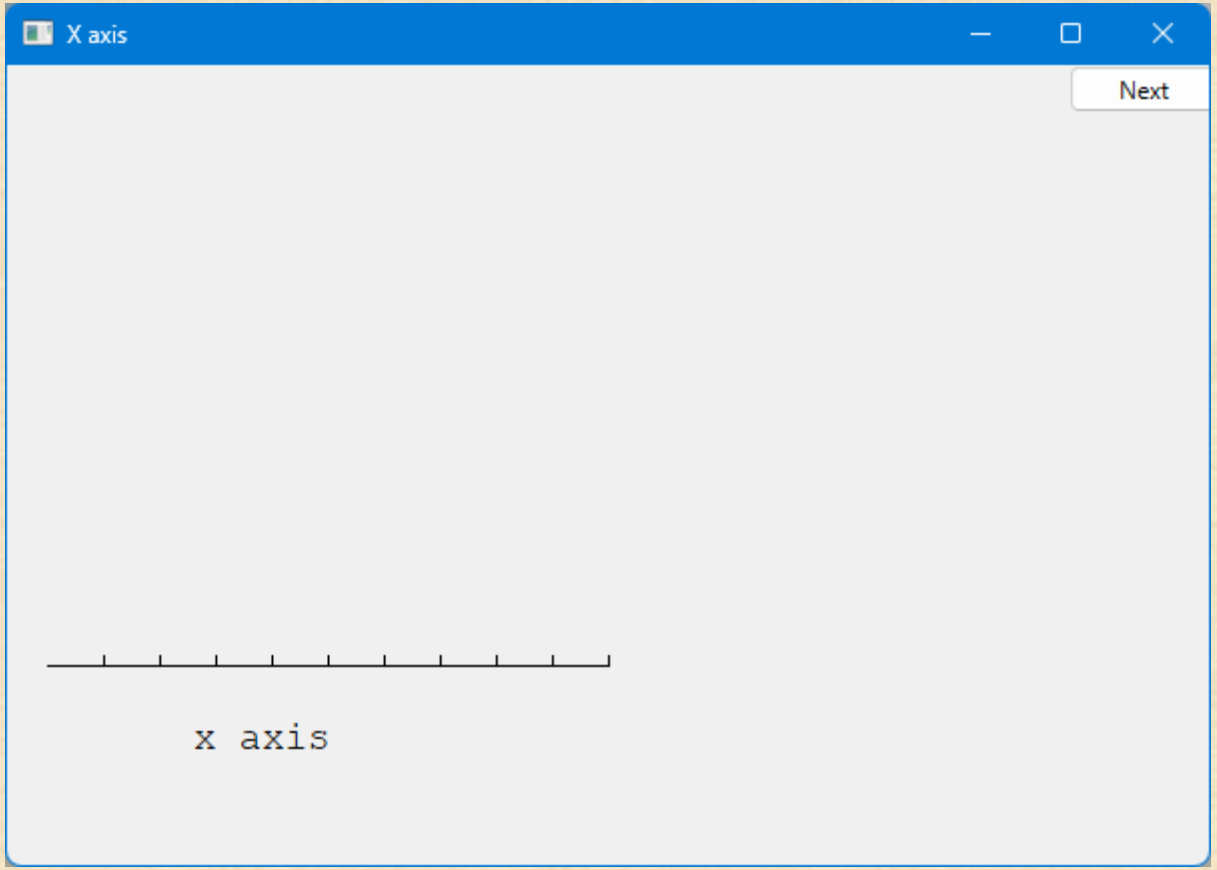
Canvas on screen



Add an x axis

```
Axis xa {Axis::x, Point{20,300}, 280, 10, "x axis"};

    // an axis is a kind of Shape
    // Axis::x means horizontal
    // starting at (20,300)
    // 280 pixels long
    // 10 "notches" ("tick marks")
    // label the axis  "x axis"

win.attach(xa);       // attach axis xa to the window
win.set_label("X axis");    // re-label the window
win.wait_for_button();      // display!
```

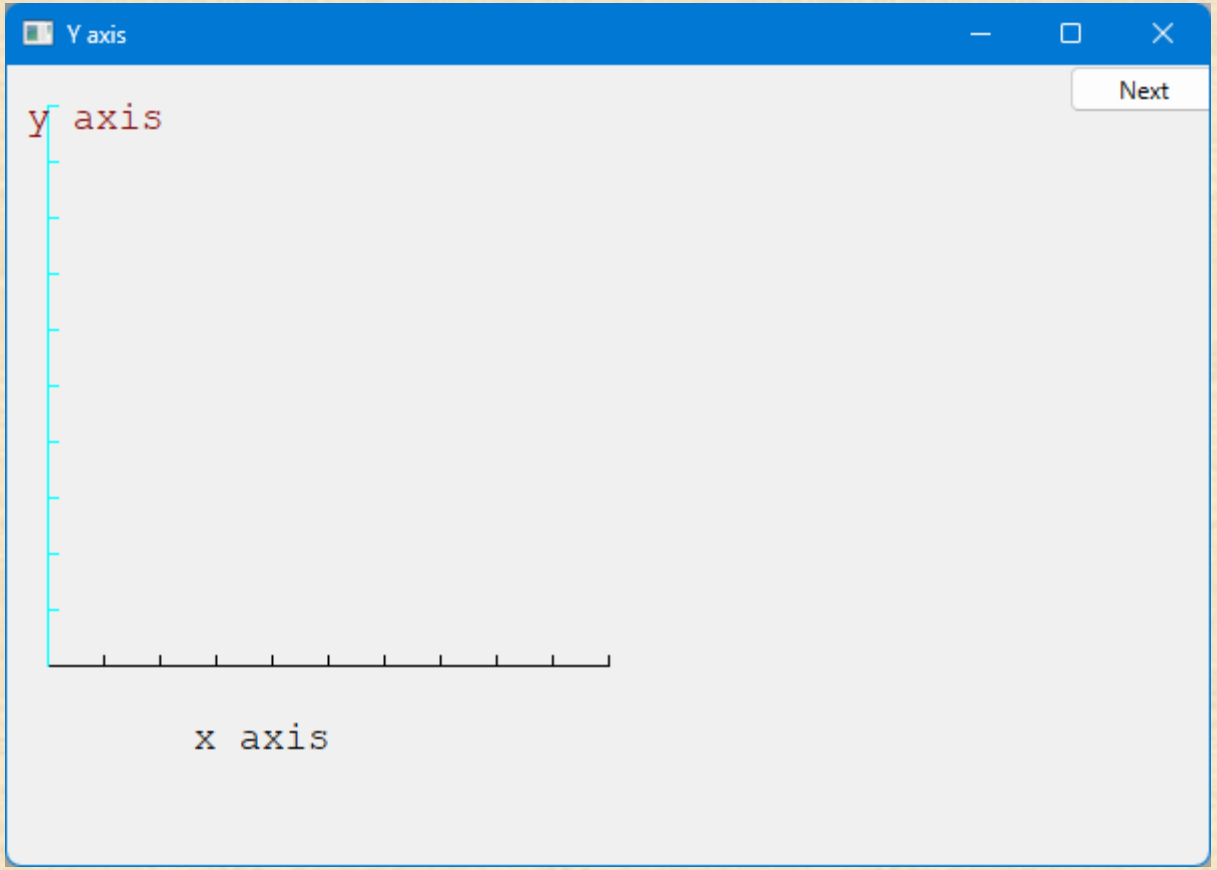
An x axis



Add a y axis

```
win.set_label("Canvas #3");

Axis ya {Axis::y, Point{20,300}, 280, 10, "y axis"};
ya.set_color(Color::cyan);                  // choose a color
  for the axis
ya.label.set_color(Color::dark_red);          // choose a
  color for the text

win.attach(ya);
win.set_label("Y axis");    // re-label the window
win.wait_for_button();
```

Add a Y-axis (colored)

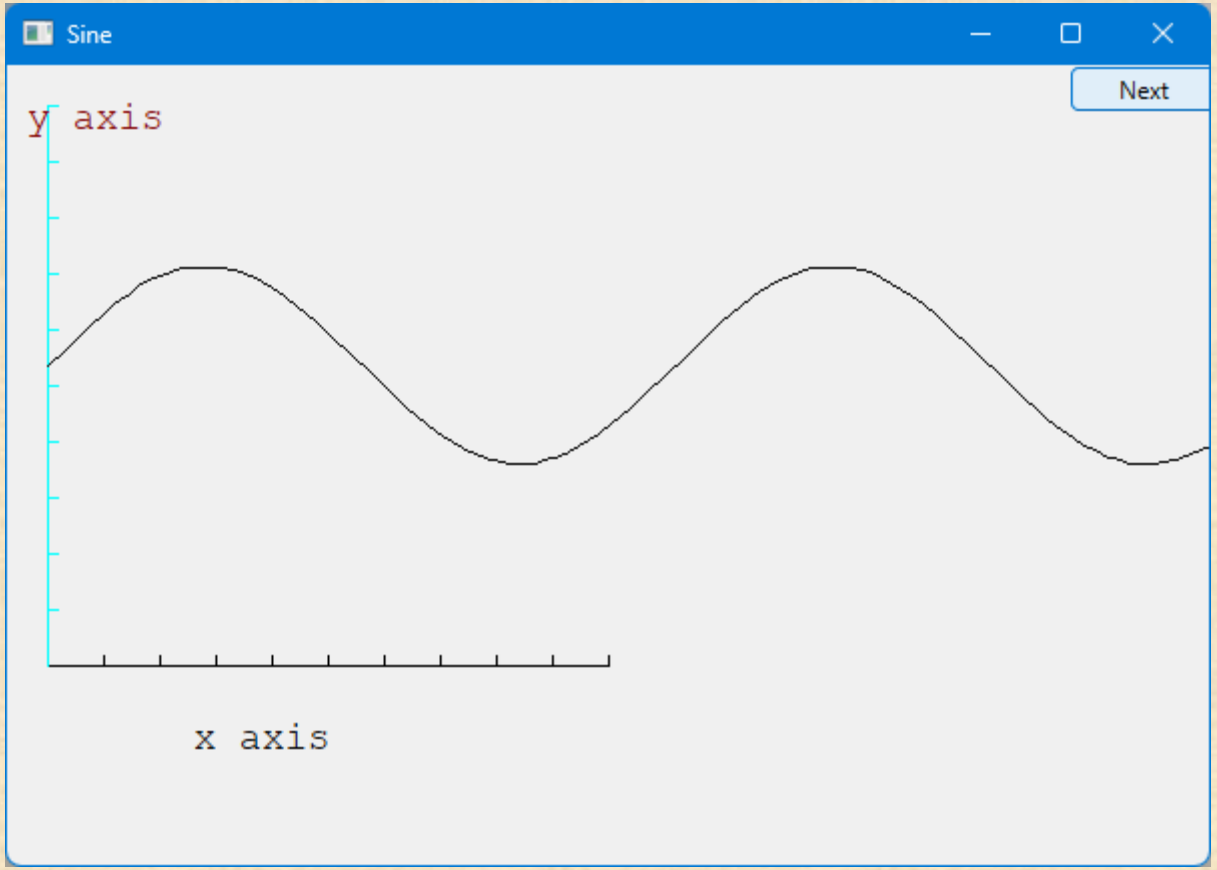


Yes, it's ugly, but this is a programming course, not a graphics design course

Add a sine curve

```
Function sine {sin,0,100,Point{20,150},1000,50,50};
        // sine curve
        // plot sin() in the range [0:100)
        // with (0,0) at (20,150)
        // using 1000 points
        // scale x values *50, scale y values *50

win.attach(sine);
win.set_label("Sine");
win.wait_for_button();
```

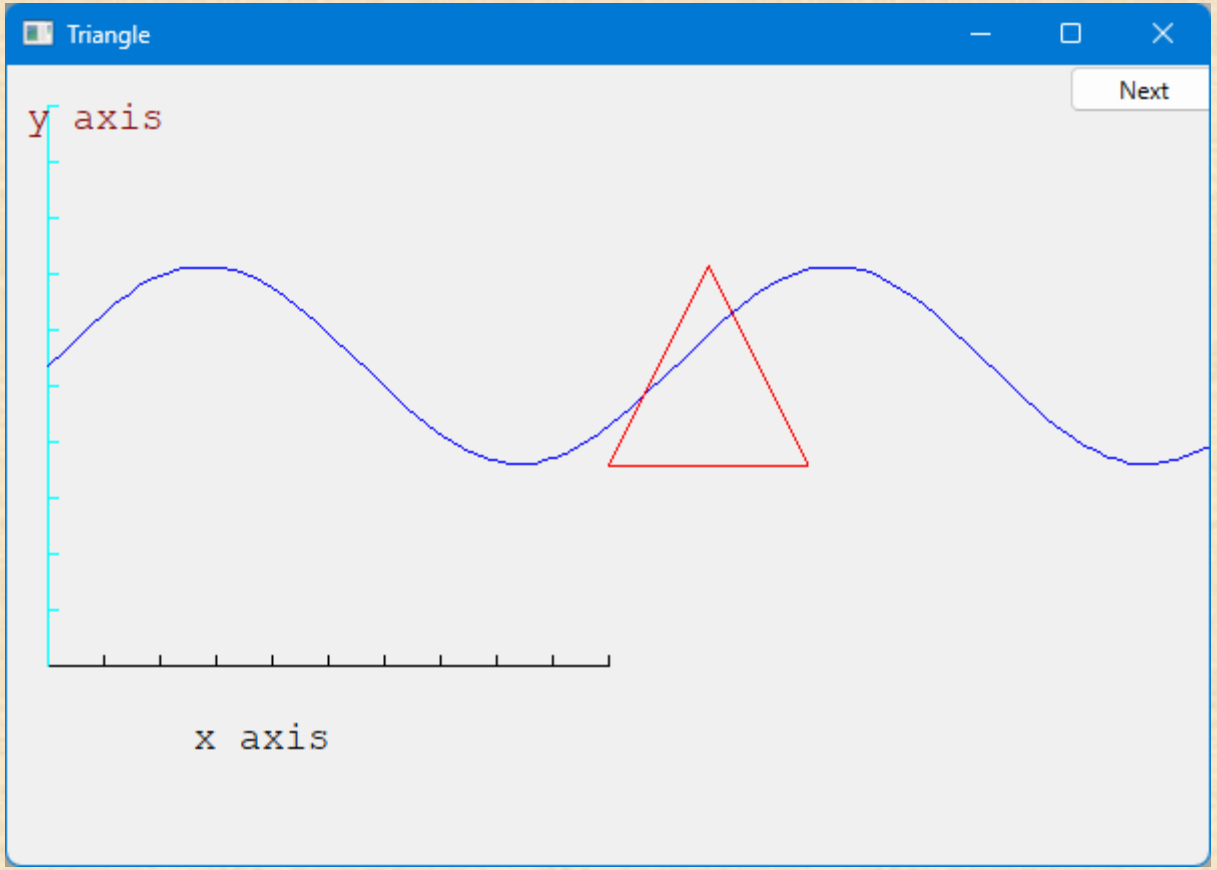
Add a sine curve



Add a polygon

```
sine.set_color(Color::blue);  // I changed my mind about
  sine's color

Polygon poly;                 // make a polygon (a kind
  of Shape)
poly.add(Point{300,200});     // three points make a
  triangle
poly.add(Point{350,100});
poly.add(Point{400,200});

poly.set_color(Color::red);   // change the color

win.attach(poly);
win.set_label("Triangle");
win.wait_for_button();
```

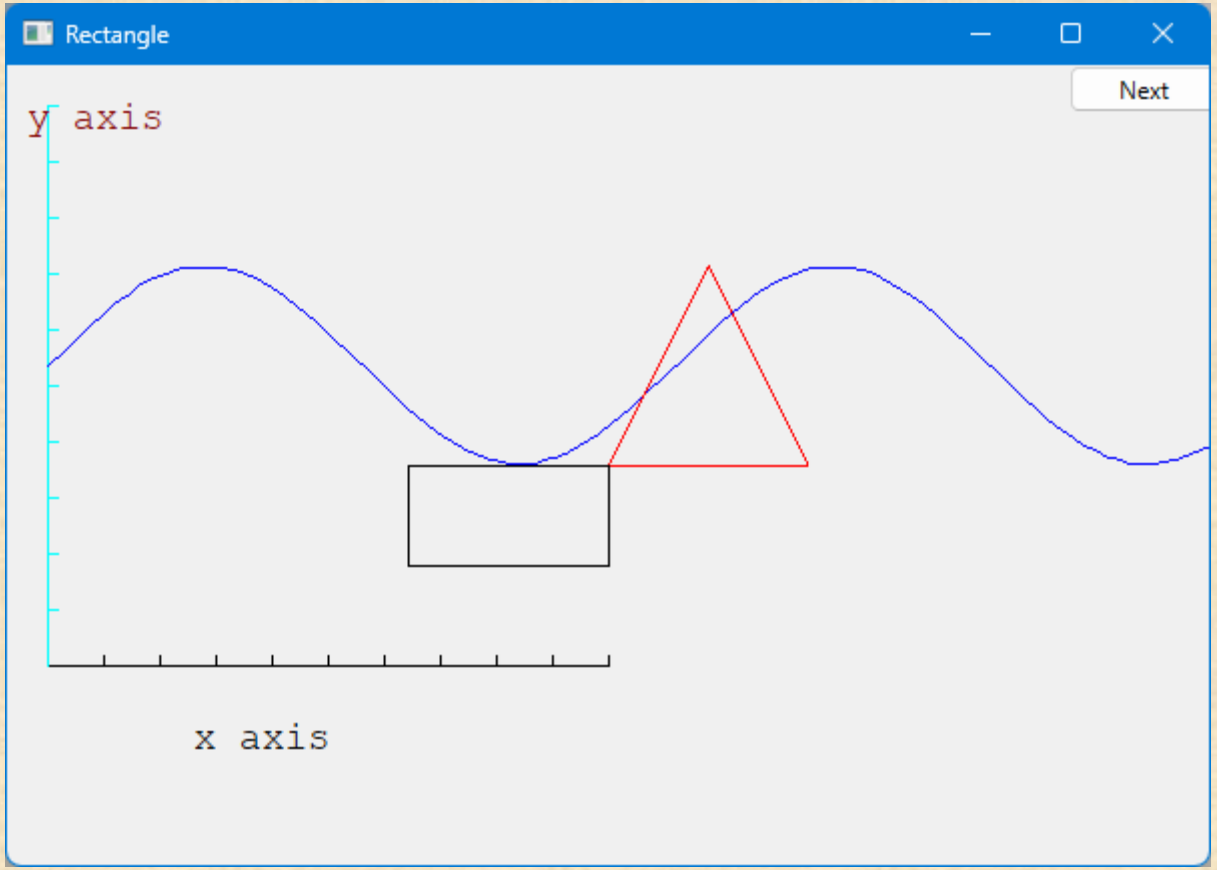
Add a triangle (and color the sine)



Add a rectangle

```
Rectangle r {Point{200,200}, 100, 50};   // top left point, width, height

win.attach(r);
win.set_label("Rectangle");
win.wait_for_button();
```

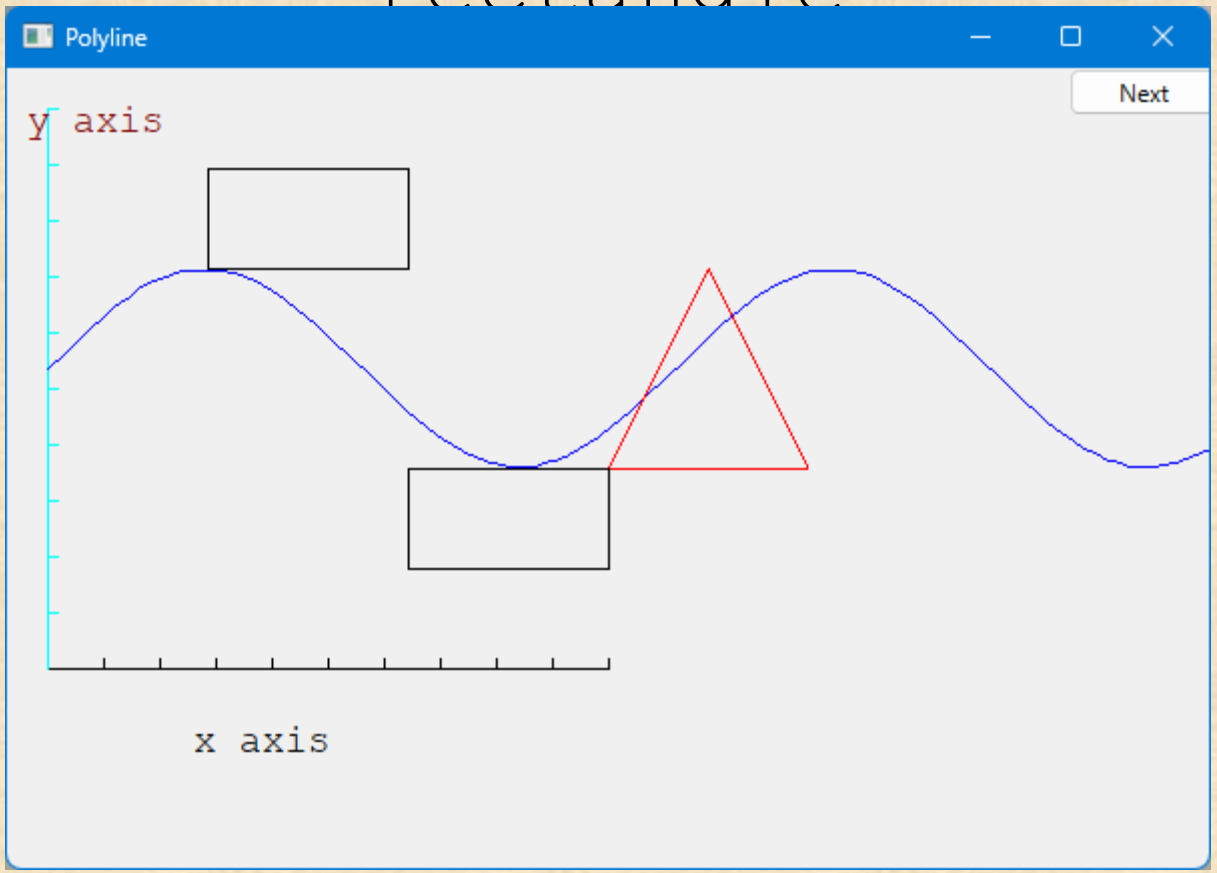
Add a rectangle



Add a shape that looks like a rectangle

```
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});

win.set_label("Polyline");
win.attach(poly_rect);
win.wait_for_button();

// a Closed_polyline is a sequence of lines ending at the
  starting point
```

Add a shape that looks like a rectangle



But is it a rectangle?

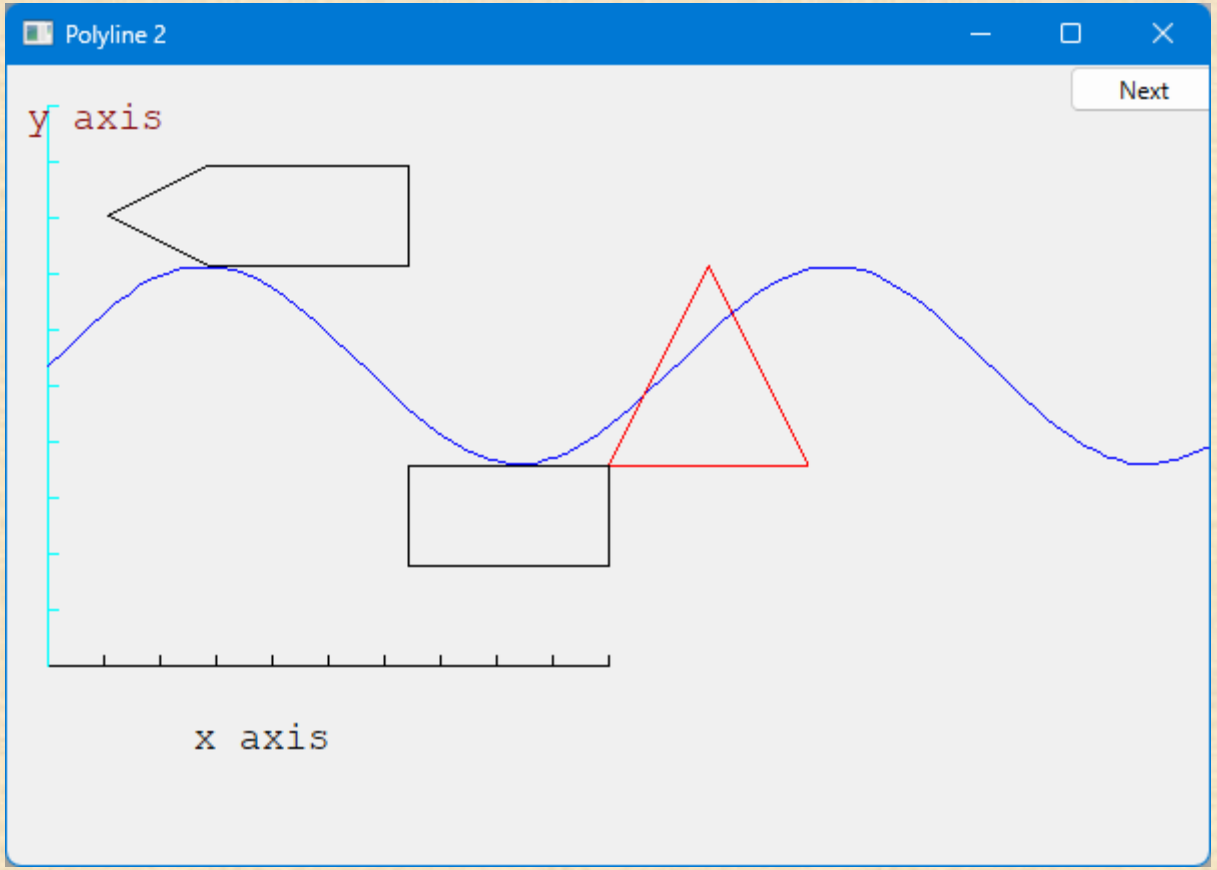
Mutate the polyline

- We can add a point

```
poly_rect.add(Point{50,75});          // now poly_rect has 5 points

win.set_label("Polyline2");
win.wait_for_button();
```

- "looking like" is not the same as "is"

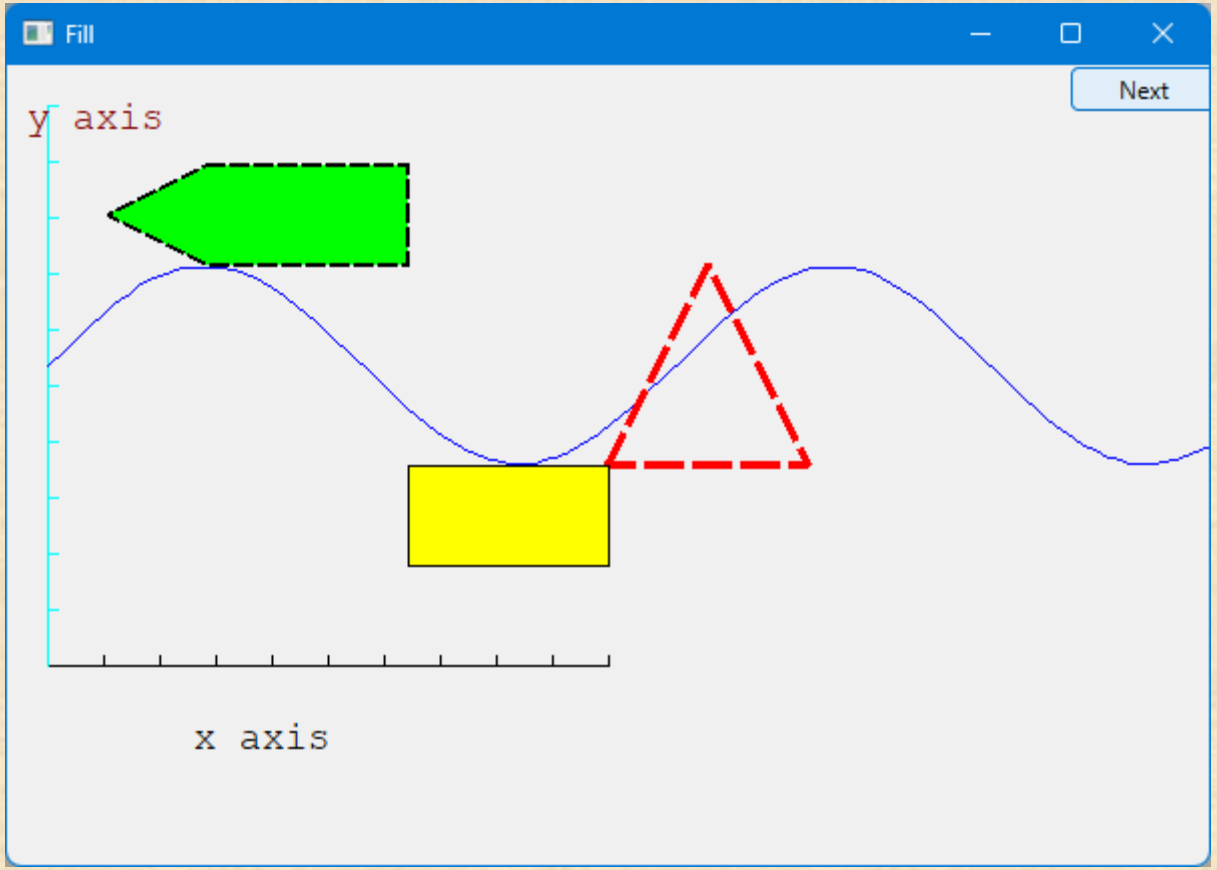
Obviously not a rectangle



Add fill

```
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});

win.set_label("Fill");
win.wait_for_button();
```

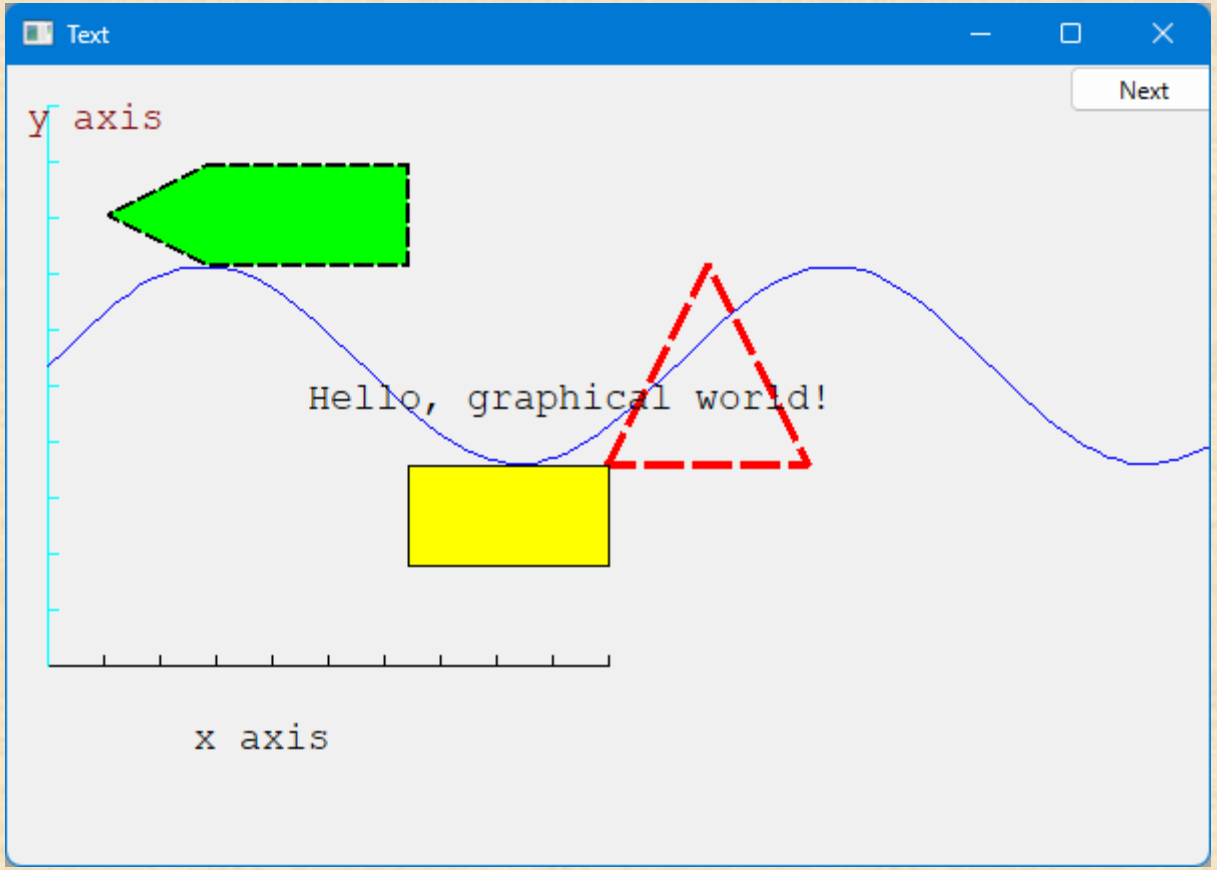
Add fill



Add text

```
Text t {Point{150,150},"Hello, graphical world!"};    // add text
                                                      // point is lower left corner on the baseline

win.set_label("Text");
win.wait_for_button();
```

Add text

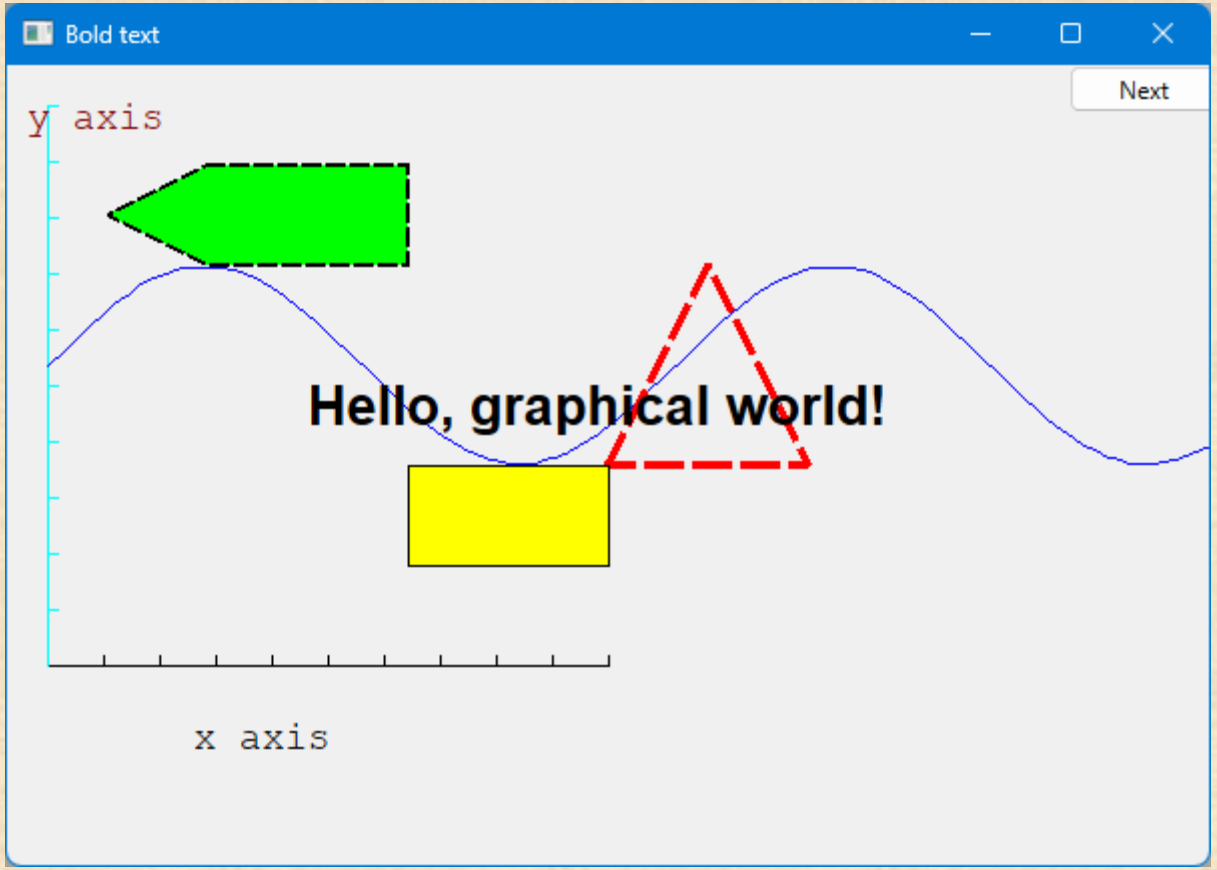


Modify text font and size

- Modify text font and size
  - **t.set_font(Font::times_bold);**
  - **t.set_font_size(20);** **//** *height in pixels*

```
win.set_label("Bold text");
win.wait_for_button();
```

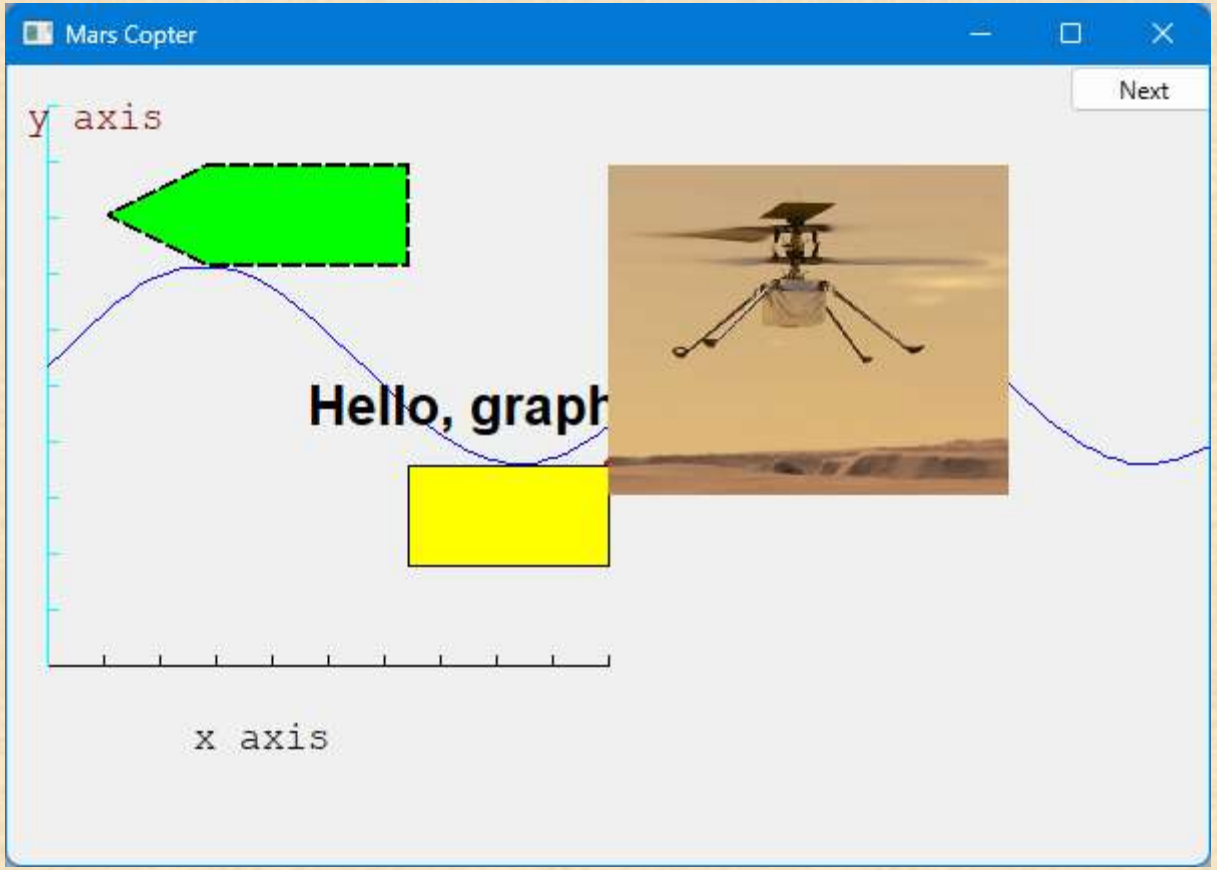
Text font and size



Add an image

```
Image copter {Point{100,50},"mars_copter.jpg"};      // open an image file

win.attach(copter);
win.set_label("Mars copter");
win.wait_for_button();
```

Add an image

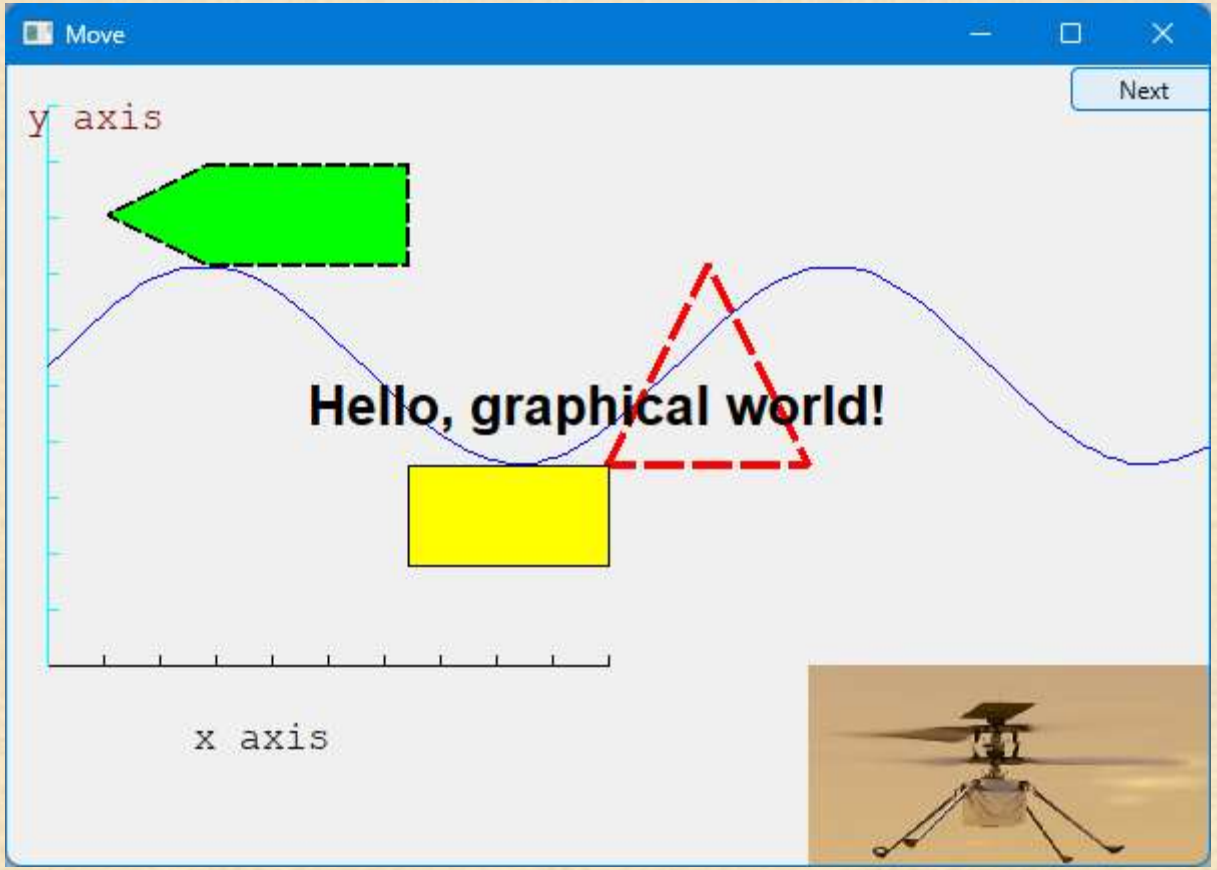


Oops!

- The image obscures the other shapes
  - Move it a bit out of the way ("fly it")

```
copter.move(100,250);  // move 100 pixels to the right (-100 moves left)
                       // move 250 pixels down (-250 moves up)
win.set_label("Move");
win.wait_for_button();
```

Move the image

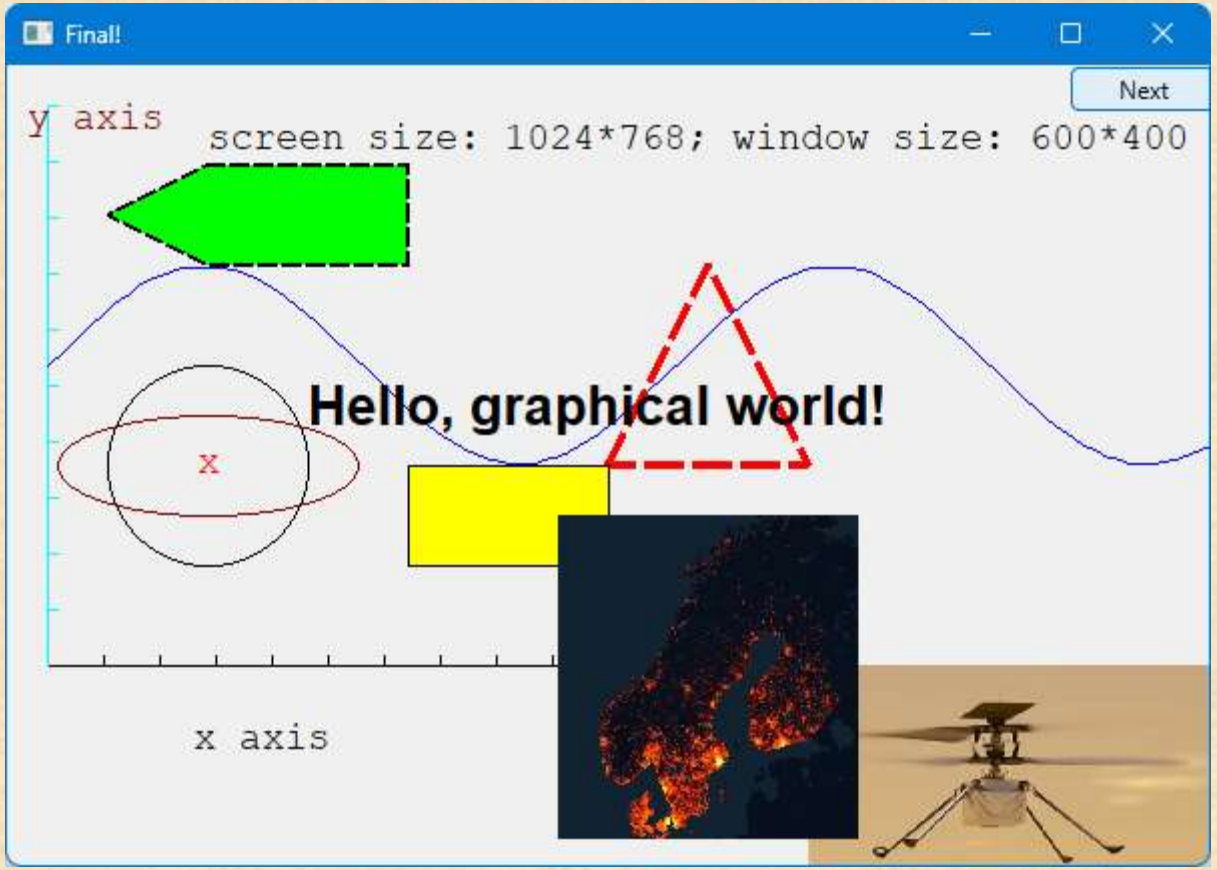


Note how the parts of a shape that don't fit in the window are "clipped" away

Add more shapes and more text

```
Circle c {Point{100,200},50};       // center, radius

Ellipse e {Point{100,200}, 75,25};  // center, horizontal radius,
  vertical radius
e.set_color(Color::dark_red);

Mark m {Point{100,200},'x'}; radius
m.set_color(Color::red);

ostringstream oss;
oss << "screen size: " << x_max() << "*" << y_max()
     << "; window size: " << win.x_max() << "*" << win.y_max();
Text sizes {Point{100,20},oss.str()};

Image scan {Point{275,225}, "scandinavia.jfif"};
scan.scale(150,200);            // scale the image to taste

// … attach all new objects …
win.set_label("Final");
win.wait_for_button();
```

Add more shapes and more text



Boiler plate

```
#include "Graph.h"                // header for graphs
#include “Simple_window.h"       // header containing window interface

int main ()
try
{
 // … the  main part of your code …
}
catch(exception& e) {
 cerr << "exception: " << e.what() << '\n';
 return 1;
}
catch (...) {
 cerr << "Some exception\n";
 return 2;
}
```

Primitives and algorithms

- The demo shows the use of library primitives
  - Just the primitives
  - Just the use
- Typically, what we display is the result of
  - an algorithm
  - reading data

- Next lectures
  - 11: Graphics Classes
  - 12: Graphics Class Design
  - 13: Graphing Functions and Data
  - 14: Graphical User Interfaces