USTC

Chapter 16
Graphical User Interfaces

王子磊 (Zilei Wang)
Email: zlwang@ustc.edu.cn
http://vim.ustc.edu.cn

Overview

- 视角
  - I/O 方式
  - GUI
  - 软件层次
- GUI 示例
- GUI 代码
  - 回调函数

I/O 方式

- 使用控制台输入输出
  - 对于专业技术而言，这是一种强有力的方式
  - 命令行接口
  - 菜单驱动接口
- 图形用户界面 GUI
  - 使用一个 GUI 库
  - 非常匹配 windows/Mac 应用的感觉
  - 当你需要点击、拖放、WYSIWYG时
  - 事件驱动的程序设计
  - Web浏览器——它是一种特殊GUI应用程序
    - 需要 HTML / 一种脚本语言
    - 对于远程访问需求是比较理想的

一般的GUI任务

- Titles/ Texts
  - 名称
  - 提示字符
  - 用户指令
- Fields / Dialog boxes
  - 输入
  - 输出
- Buttons
  - 让用户触发动作
  - 让用户从一个可选集合中进行选择
    - e.g. yes/no, blue/green/red, etc.

一般的GUI任务

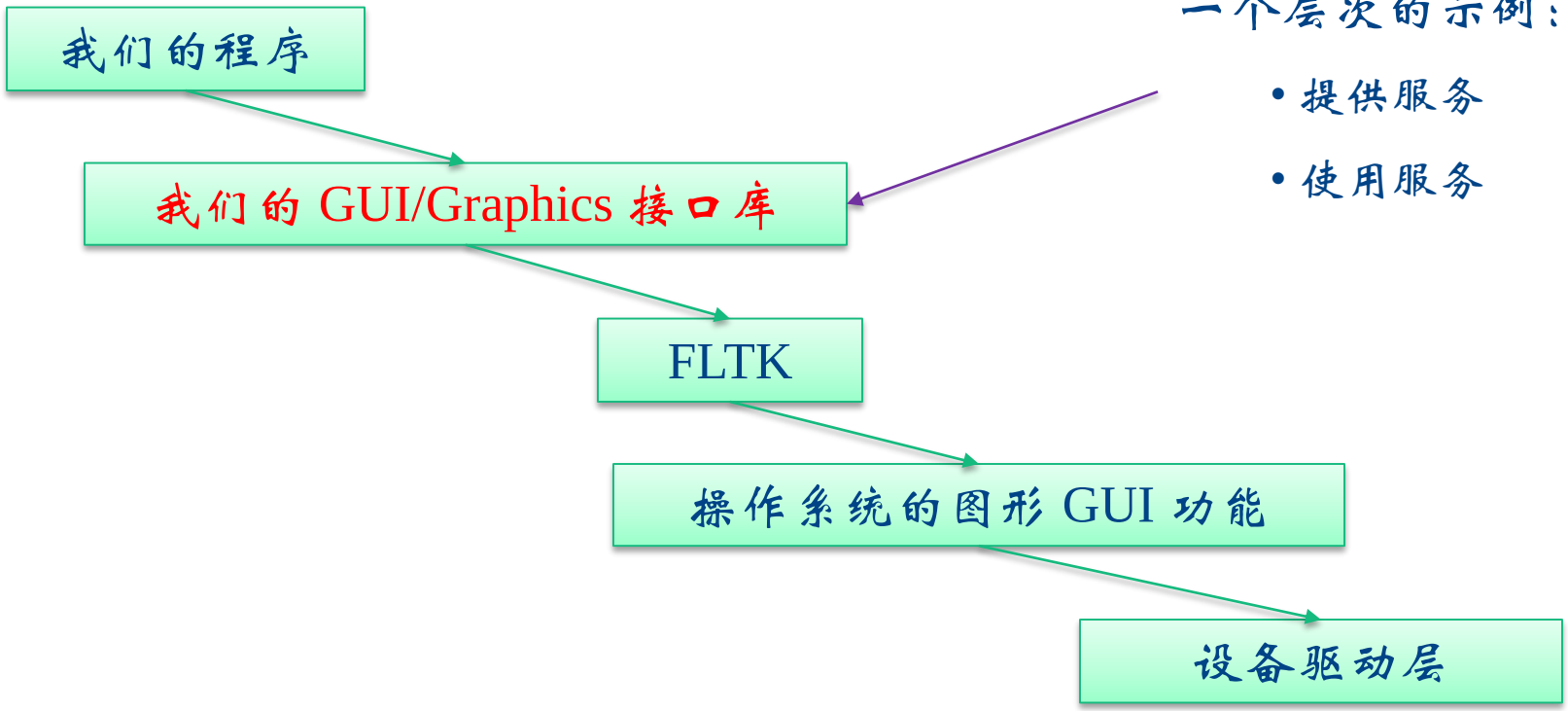
- 显示结果
  - 图形
  - 文本和数字
- 生成比较合理的窗口
  - Style 和 color
    - 我们此处的窗口在不同的系统上看起来可能是不同的
- 更多...
  - 跟踪鼠标
  - 拖放
  - 自由绘制

GUI

- 从程序设计的角度看GUI，它基于以下两个技术
  - 面向对象编程
    - 在组织程序方面，包括共用接口和共用动作
  - 事件
    - 连接一个事件(如鼠标单击)和一个程序动作
- GUI是一种I/O形式
  - 应用程序的主要逻辑和I/O相互分离是软件设计的主要观点之一

软件层次

- 当我们开发软件时，通常以已有的代码为基础



一个层次的示例：

- 提供服务
- 使用服务

GUI 示例

- 一个窗口，拥有
  - 两个 Buttons、 两个 In_boxes、一个 Out_box

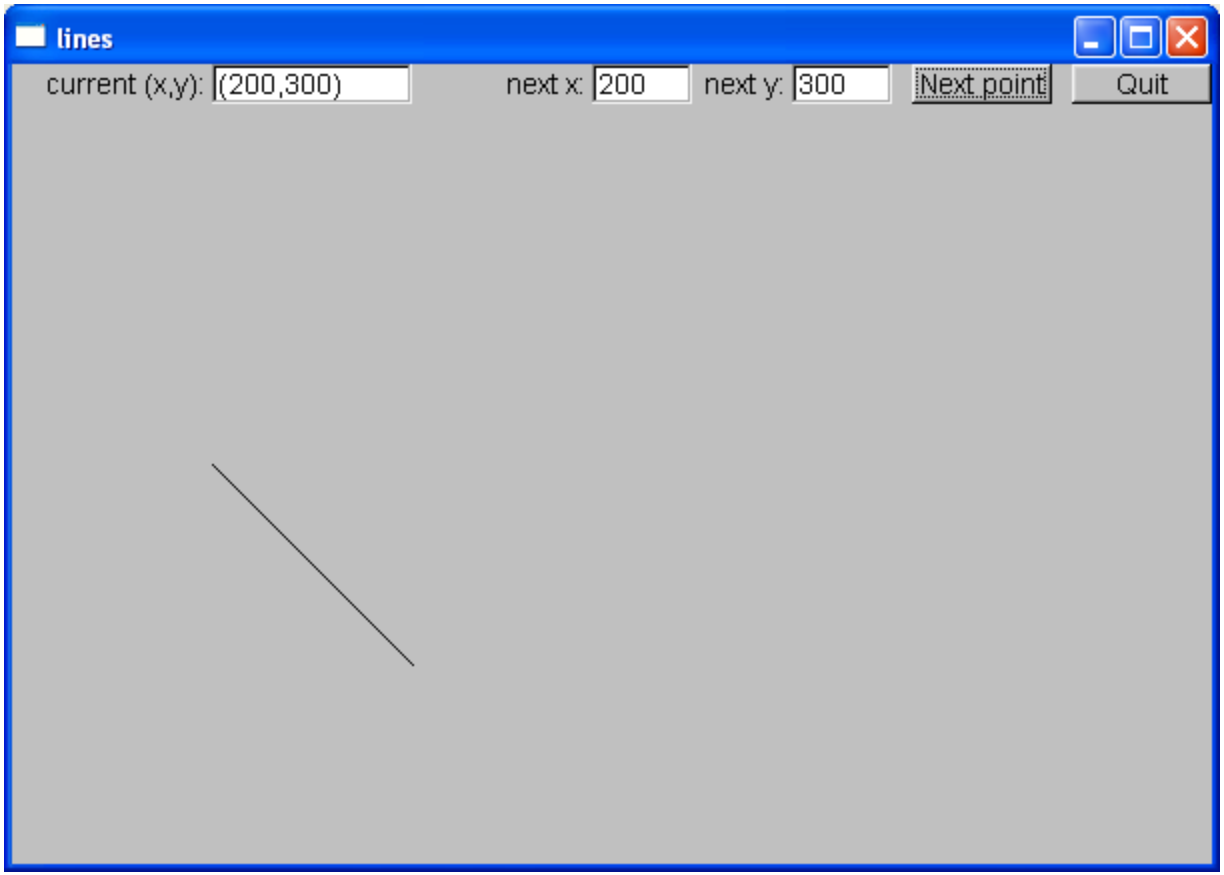
GUI 示例

- 在 In_boxes 中输入一个点

GUI 示例

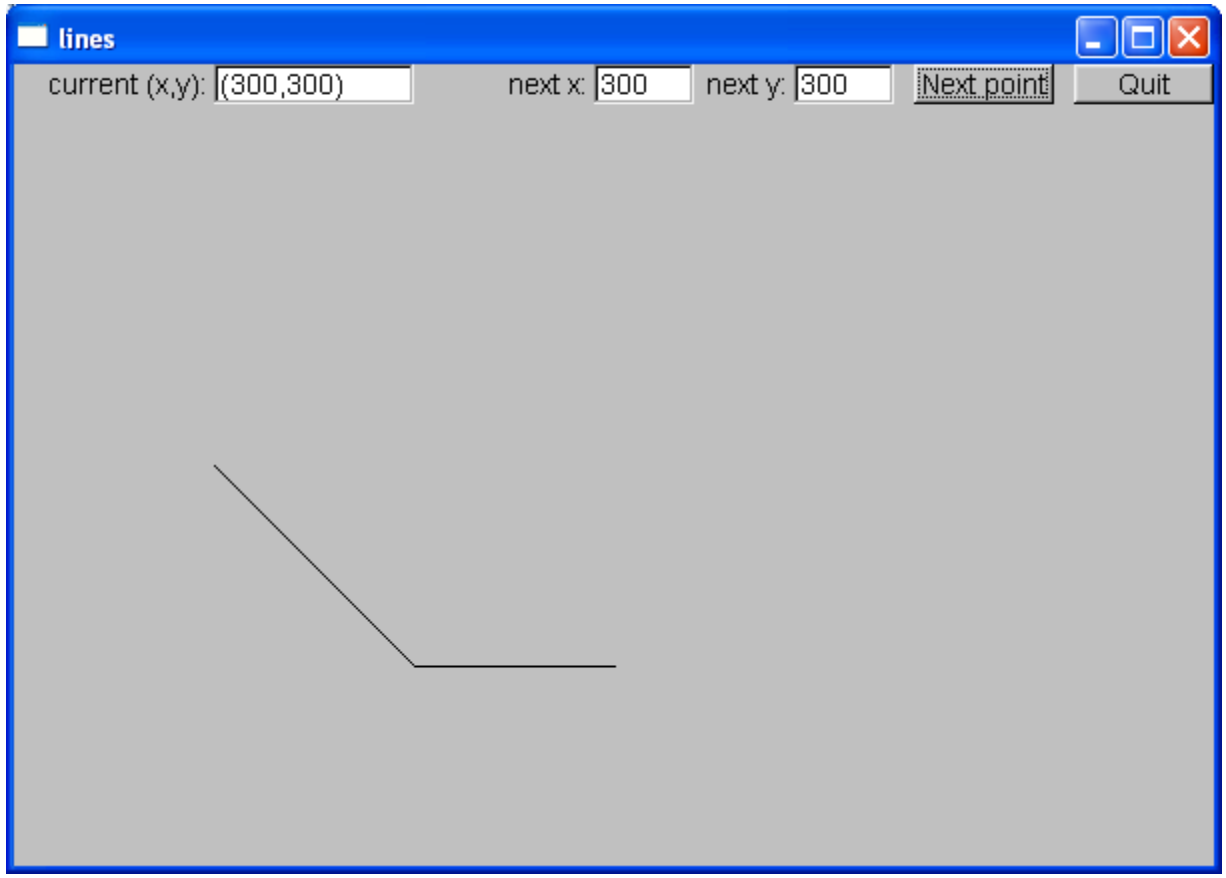
- 当你点击**next point** 后，输入的点变为当前点current (x,y)，并显示在 **Out_box** 中

GUI 示例



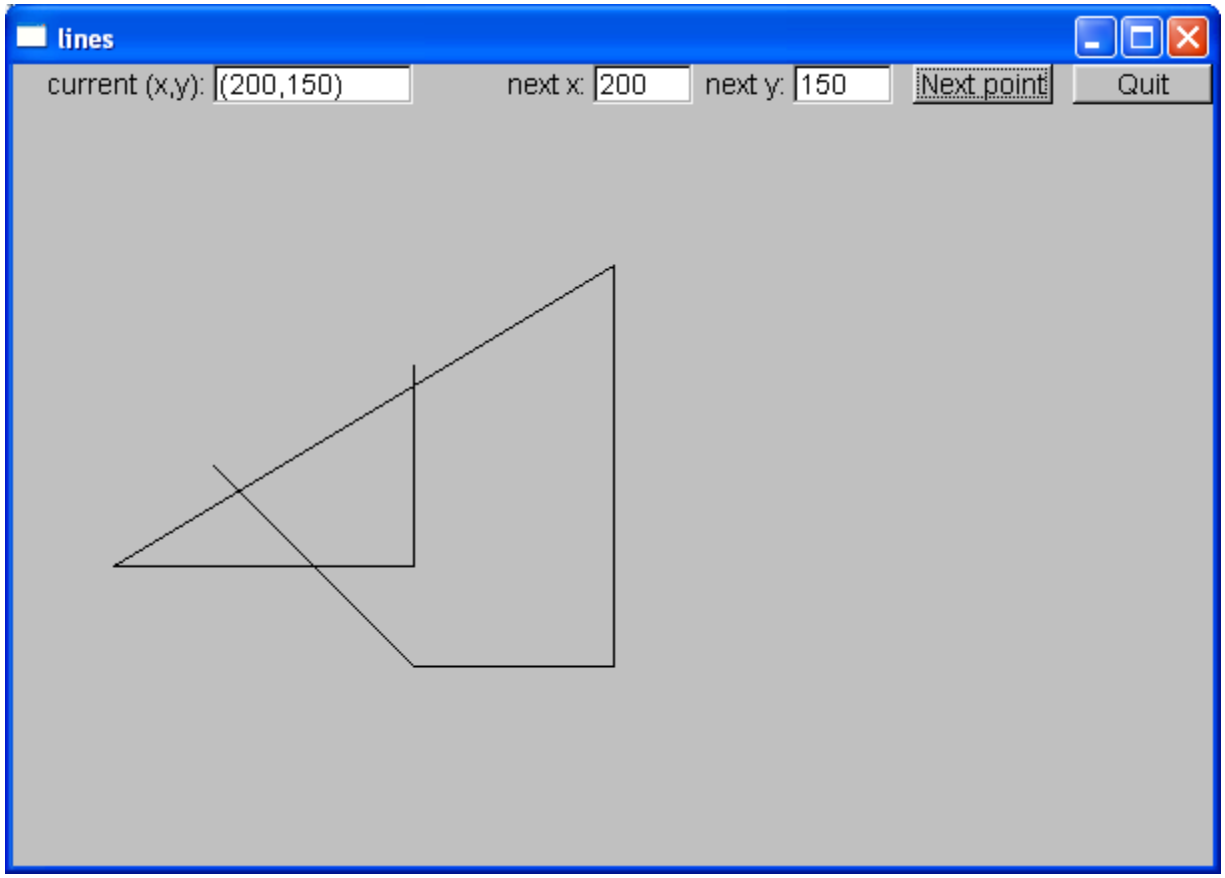
- 添加另外一个点以后，就会显示一条线

GUI 示例



- 三个点绘制两条线
  - 显然，我们是在绘制一个polyline

GUI 示例



- 如此能够循环执行，直到点击 Quit 为止

那么，需要做什么以及怎么做？

- 窗口中包括 buttons, input boxes 和 outbox
  - 如何定义一个窗口？
  - 如何定义一个按钮？
  - 如何定义输入和输出框？
- 单击按钮及其相关的处理动作
  - 如何编码这种动作？
  - 如何连接我们的代码和对应的按钮？
- 在输入框中键入
  - 如何获取这些值到我们的代码中？
  - 如何从一个string转换为我们需要的数字？
- 从输出框中输出
  - 如何获取输出相关值？
- 在窗口中绘制线形
  - 如何存储线形？
  - 如何绘制它们？

匹配映射

- 我们将前面的思想转换到传统的图形GUI上，即此处的FLTK

定义类 Lines_window

```
struct Lines_window : Window          // Lines_window inherits from Window
{
    Lines_window(Point xy, int w, int h, const string& title); // declare constructor
    Open_polyline lines;

private:
    Button next_button;               // declare some buttons – type Button
    Button quit_button;
    In_box next_x;                    // declare some i/o boxes
    In_box next_y;
    Out_box xy_out;

    void next();                      // what to do when next_button is pushed
    void quit();                      // what to do when quit_botton is pushed

    static void cb_next(Address, Address window); // callback for next_button
    static void cb_quit(Address, Address window);  // callback for quit_button
};
```

按钮实际处理的实现代码

GUI 示例

- 一个窗口，拥有
  - 两个 Buttons、 两个 In_boxes、一个 Out_box

Lines_window 的构造函数

```
Lines_window::Lines_window(Point xy, int w, int h, const string& title)
    :Window(xy,w,h,title),
        // construct/initialize the parts of the window:
                    // location          size      name          action
    next_button(Point(x_max()-150,0), 70, 20, "Next point", cb_next),
    quit_button(Point(x_max()-70,0), 70, 20, "Quit", cb_quit),      // quit button
    next_x(Point(x_max()-310,0), 50, 20, "next x:"),                // io boxes
    next_y(Point(x_max()-210,0), 50, 20, "next y:"),
    xy_out(Point(100,0), 100, 20, "current (x,y):")
{
    attach(next_button);                // attach the parts to the window
    attach(quit_button);
    attach(next_x);
    attach(next_y);
    attach(xy_out);
    attach(lines);                      // attach the open_polylines to the window
}
```

Widgets, Buttons 和回调

- 构件 (Widget) 是窗口中具有关联动作的东东
- 按钮 (Button) 是一个Widget，它在屏幕上是一个带有标签的矩形区域
  - 当你单击它时，一个回调函数会被触发
- 回调 (Callback) 连接了按钮和一个或多个相关函数——实际执行的动作

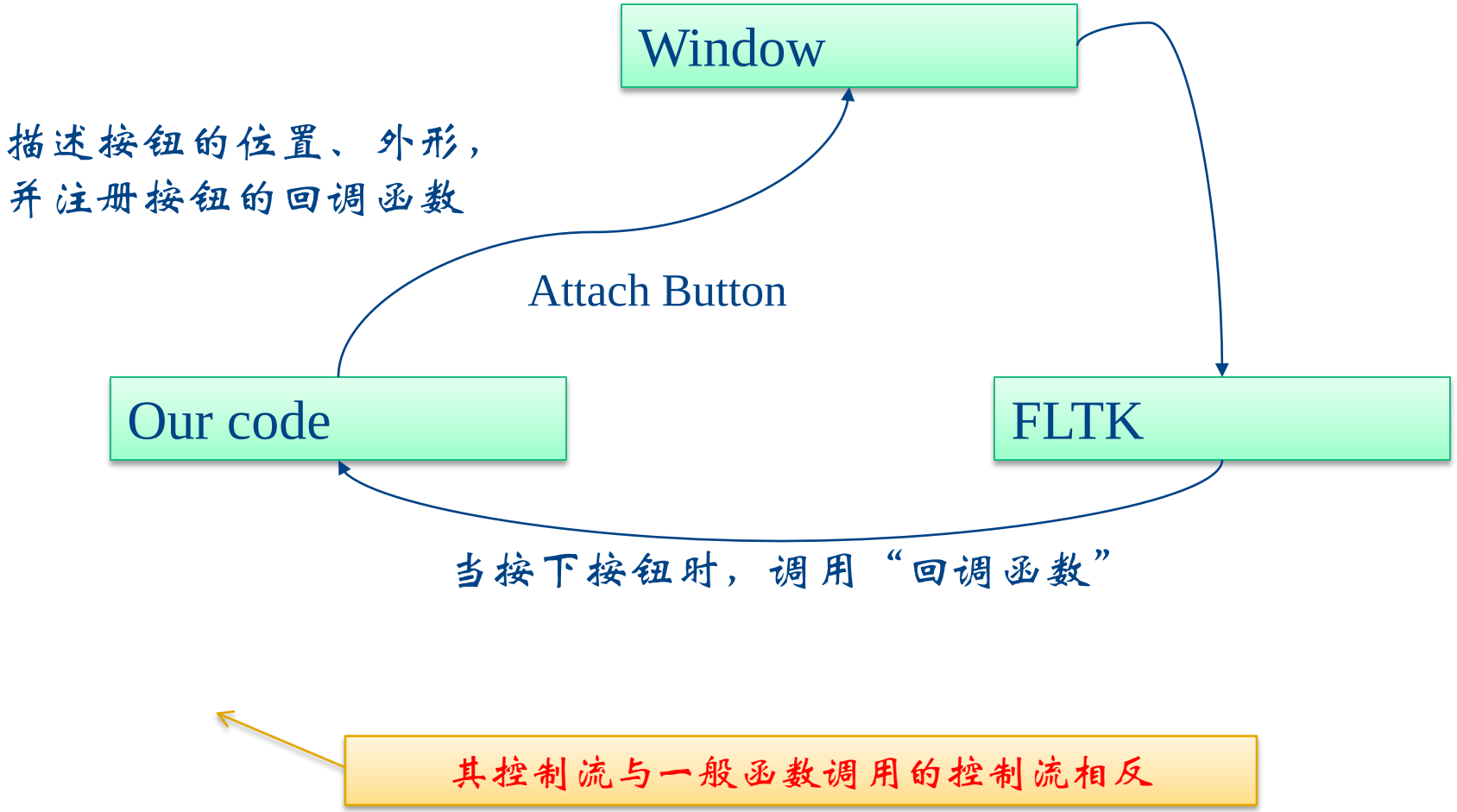
Widgets, Buttons 和回调

// A widget is something you see in the window
// which has an action associated with it

// A Button is a Widget that displays as a labeled rectangle on the screen;
// when you click on the button, a Callback is triggered
// A Callback connects the button to some function

```
struct Button : Widget {
    Button(Point xy, int w, int h, const string& s, Callback cb)
        :Widget(xy,w,h,s,cb) { }
};
```

它是如何工作的？

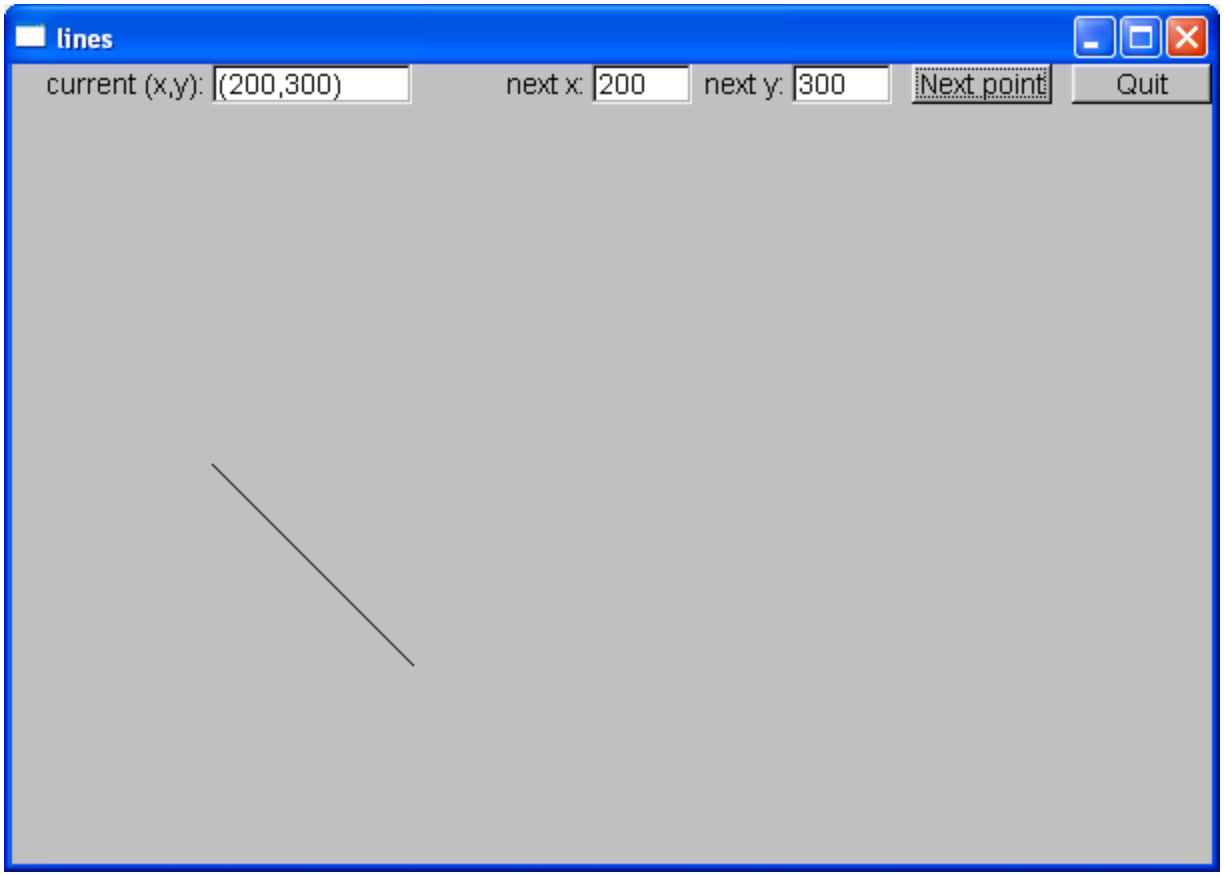

Window
描述按钮的位置、外形，
并注册按钮的回调函数
Attach Button
Our code
FLTK
当按下按钮时，调用“回调函数”
其控制流与一般函数调用的控制流相反

Widget

- 它是Windows 和 X 窗口系统的一个基本概念
  - 基本上，你在屏幕上看到的以及做的都是一个widget
  - 又称之为控件 (“control”)

```
struct Widget {
    Widget(Point xy, int w, int h, const string& s, Callback cb)
      :loc(xy), width(w), height(h), label(s), do_it(cb)
    { }
    // ... connection to FLTK ...
};
```

GUI 示例



- 添加另外一个点以后，就会显示一条线

Button

- 一个按钮 Button 是一个 Widget，同时满足
  - 显示时，是屏幕上具有标签的一个矩形区域
  - 单击时，会触发一个回调函数

```
struct Button : Widget {
    Button(Point xy, int w, int h, const string& s, Callback cb)
      :Widget(xy,w,h,s,cb) { }
};
```

Callback

- 回调是我们与系统交互的一种接口形式
  - 在大多数GUI中，将函数连接到widget是混乱的
  - 此处的回调相对简单，但
    - “系统”并不十分“了解”C++
    - 其风格（混乱的方式）源自于C/汇编设计的系统
    - 大多数系统是使用多种语言的
      - 这是说明如何跨语言设计的一个例子
  - 回调函数将我们的程序从系统转移到C++的世界中(我们熟悉和控制范围内)

```
void Lines_window::cb_quit(Address, Address pw)
    // Call Lines_window::quit() for the window located at address pw
{
    reference_to<Lines_window>(pw).quit();    // now call our function
}
```

映射一个地址为一个引用(对象类型在地址中)
下一章会具体解释

我们的执行代码

```
// The action itself is simple enough to write

void Lines_window::quit()
{
    // here we can do just about anything with the Lines_window
    hide();          // peculiar FLTK idiom for "get rid of this window"
}
```

next 函数

```
// our action for a click ("push") on the next button
void Lines_window::next()
{
    int x = next_x.get_int();
    int y = next_y.get_int();

    lines.add(Point(x,y));

    // update current position readout:
    stringstream ss;
    ss << '(' << x << ',' << y << ')';
    xy_out.put(ss.str());

    redraw();   // now redraw the screen
}
```

In_box

```
// An In_box is a widget into which you can type characters
// It's "action" is to receive characters

struct In_box : Widget {
    In_box(Point xy, int w, int h, const string& s)
        :Widget(xy,w,h,s,0) { }
    int get_int();
    string get_string();
};

int In_box::get_int()
{
    // get a reference to the FLTK FL_Input widget:
    Fl_Input& pi = reference_to<Fl_Input>(pw);
    // use it:
    return atoi(pi.value());   // get the value and convert
                               // it from characters (alpha) to int
}
```

总结

- 我们已经看到
  - 按钮上的动作
  - 交互I/O
    - 文本输入
    - 文本输出
    - 图形输出
- 没有提及的内容
  - Menu (参见16.7)
  - Window 和 Widget (参见 Appendix E)
  - 跟踪鼠标的相关内容
    - Dragging
    - Hovering
    - Free-hand drawing

- 其他没有提及的内容，请感兴趣的同学自己查看
  - 细节是与工具相关的，已经偏离了我们的核心——程序设计

Next

- 接下来的三章将说明如何使用底层的语言特性来实现标准 vector