Chapter 12
A display model

Bjarne Stroustrup
www.stroustrup.com/Programming

Abstract

This lecture presents a display model (the output part of a GUI), giving examples of use and fundamental notions such as screen coordinates, lines, and color. Some examples of shapes are Lines, Polygons, Axis, and Text.

Overview

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 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

The resulting screen

Graphics/GUI libraries

• You’ll be using a few interface classes we wrote
  – Interfacing to a popular GUI toolkit
    • GU = Graphical User Interface
    • FLTK: www.fltk.org // Fast Light Tool Kit
  – Installation, etc.
    • see Appendix D and ask instructor/friend
      – FLTK
      – Our GUI and graphics classes
      – Project settings

• This model is far simpler than common toolkit interfaces
  – The FLTK (very terse) documentation is 370 pages
  – Our interface library is <20 classes and <500 lines of code
  – You can write a lot of code with these classes
  – And what you can build on them

Graphics/GUI libraries (cont.)

• The code is portable
  – Windows, Unix, Mac, 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 re-size a window (changing x_max() and y_max())

Interface classes

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

Demo code 1

```cpp
// 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
using namespace Graph_lib;
// make names available

// in main():
Simple_window win(Point(100,100),600,400,"Canvas");
// screen coordinate (100,100) top left of window
// title: Canvas
win.wait_for_button(); // Display!
```

Demo code 2

```cpp
Axis xa(Axis::x, Point(20,300), 280, 10, "x axis");
// make an Axis
// an axis is a kind of Shape
// Axis::x means horizontal
// starting at (20,300)
// 280 pixels long
// 10 “notches”
// text “x axis”
win.set_label("Canvas #2");
win.attach(xa); // attach axis xa to the window
win.wait_for_button();
```
Add an X-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.wait_for_button();
```

Add a Y-axis (colored)

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

```
win.set_label("Canvas #4");
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.wait_for_button();
```

Add a sine curve

```
win.set_label("Canvas #5");
sine.set_color(Color::blue);  // I changed my mind about sine's color
Polygon poly;  // a polygon, a Polygon is a kind of Shape
  poly.add(Point(300,200));  // three points makes a triangle
  poly.add(Point(350,100));
poly.add(Point(400,200));
poly.set_color(Color::red);  // change the color
poly.set_style(Line_style::dash);  // change the line style
win.attach(poly);
win.wait_for_button();
```
Add a triangle (and color the curve)

Demo code 6

```cpp
win.set_label("Canvas #6");
Rectangle r(Point(200,200), 100, 50);
win.attach(r);
win.wait_for_button();
```

Add a rectangle

Demo code 6.1

- Add a shape that looks like a rectangle

```cpp
Closed_polyline poly_rect;
poly_rect.add(Point(100,50));
poly_rect.add(Point(100,100));
poly_rect.add(Point(200,100));
poly_rect.add(Point(200,50));
win.set_label("Canvas #6.1");
```

Add a shape that looks like a rectangle

Demo code 6.2

- We can add a point
  ```cpp
poly_rect.add(Point(50,75)); // now poly_rect has 5 points
```
  ```cpp
  win.set_label("Canvas #6.2");
  ```
- “looking like” is not the same as “is”
Obviously a polygon

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("Canvas #7");

Add fill

Demo Code 8

Text t(Point(100,100),"Hello, graphical world!");  // add text
win.set_label("Canvas #8");

Add text

Demo Code 9

- Modify text font and size
  t.set_font(Font::times_bold);
  t.set_font_size(20);
Add an image

```cpp
Image ii(Point(100,50),"image.jpg"); // open an image file
win.attach(ii);
win.set_label("Canvas #10");
```

Oops!

- The image obscures the other shapes
- Move it a bit out of the way

```cpp
ii.move(100,200);
win.set_label("Canvas #11");
win.wait_for_button();
```

Demo Code 12

```cpp
Circle c(Point(100,200),50);
Ellipse e(Point(100,200),75,25);
e.set_color(Color::dark_red);
Mark m(Point(100,200),'x');
ostringstream oss;
oss << "screen size: " << x_max() << "*" << y_max();
oss << " window size: " << win.x_max() << "*" << win.y_max();
Text sizes(Point(100,20),oss.str());
Image cal(Point(225,225),"snow_cpp.gif"); // 320*240 pixel gif
cal.set_mask(Point(40,40),200,150); // display center of image
win.set_label("Canvas #12");
win.wait_for_button();
```
Add shapes, more text

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
- 13: Graphics Classes
- 14: Graphics Class Design
- 15: Graphing Functions and Data
- 16: Graphical User Interfaces

Boiler plate
```c++
#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;
}
```