Using Graphics in the First Year of Programming with C++

Dr. Thomas E. Gibbons
CS/CIS Department
College of St. Scholastica
Duluth, MN
tgibbons@css.edu

Abstract

Today, a graphical user interface to computers is assumed, but doing graphical programming with C++ in the first year of a programming class is relatively uncommon. While C++ is the a commonly used programming language, many students feel they are learning in a "toy" environment when their first year of programming is all text based. At the College of St. Scholastica, we have looked at a number of different methods for introducing a graphical interface into our C++ programming courses. The solution we currently use is two-part. First, we have chosen a textbook by Cohoon and Davis that includes a simple library for graphics programming called EZWindows. Second, we introduce basic Windows programming with Microsoft's Foundation Classes (MFC) programming in the second semester. This paper describes the two different programming environments and how they are used together.

Assumptions

A wide range of techniques is used in teaching the first two programming courses (CS1 and CS2). Some institutions prefer breadth first while others prefer depth first. Objects early vs. objects late, Java vs. C++. Rather than debate the pedagogy of these different techniques, this paper starts with a set of assumptions. First, CS1 and CS2 constitute a year long sequence of two four-credit courses that focus on programming only. Second, the programming environment will be C++ on a Windows operating system using a Microsoft compiler. The reasons why these assumptions are made will not be discussed; it can be assumed that they are given a priori. Taking into account these assumptions, the question is whether graphics should be used in the introductory C++ programming course.

The Challenge

Within these constraints, programming a graphical environment is difficult because C++ does not have a default graphics library. For this reason, many introductory programming textbooks and associated courses avoid doing any GUI programming. Instead, these courses rely on text-based programming. The reason for this is related to the difficulty encountered when teaching programming concepts and the time limits placed on a programming course. With time at a premium in programming courses, most instructors have trouble deciding which programming concepts should be omitted in order to devote time to learning a graphical environment. The problem is not helped by the fact that the industry standard GUI library, Microsoft Foundation Classes (MFC), has a very steep learning curve. Simple programs like those common in a CS1 course are difficult to write using MFC, requiring students to follow many interrelated steps and navigate through large program files in order to write their code. The standard "Hello World" application generated by the MFC wizard includes 10 files containing over 300 lines of code. The challenges faced when teaching MFC in CS1 and CS2 courses often result in first year programming students only writing programs using the console window with simple text input and output.

The Value-Minded Student

When today's student pays tuition to take a course, the student wants to know that the course will be worth its price. This means that students want to see the practical value a course has to offer and are less willing to accept courses stressing abstract theory over practical application. Given this type of student, it is difficult to teach a year of programming concepts without covering a single program using a graphical interface. Students see the programs they run are based on a GUI interface. It is also not uncommon for students to have contacts with people working in the computer field and these contacts will verify the need for sophisticated graphical programs.

It is easy for instructors to argue from their academic backgrounds that they know what is best for students and that theory is preferred over practice. Students often question this, including some alumni who have years of actual programming experience. The move to more student centered learning has added clout to the students' concerns. If we adopt a teaching paradigm where students have more control over what they learn, then we must accept it when students demand to know how a course's concepts can be applied. This is not to say that students are so simple minded as to ignore theoretical concepts. In the ever-changing world of technology, students quickly become convinced that they must understand concepts as well as tools, because tools become outdated quickly. It is the balance between theory and practice that is hard to achieve. The argument made here is that while teaching GUI programming in CS1 and CS2 might force you to eliminate some other concepts, the gains outweigh the losses.

The Fun of Programming

While it is hard to argue with academics about the need to make programming fun, it is an issue that needs to be addressed. One reason for making programming fun is the many students who are not entirely committed to a computer major. It is not uncommon for a single course to turn students off from a major, especially those students who were not entirely committed to the major to begin with. But beyond stemming attrition in our major, many programmers remember fondly some of their first "cool" programs and for many people these were programs that involved graphics. Maybe it was a program that displayed a fractal image, or a simple game, or a bouncing rectangle. There is something about being able to use a programming language to manipulate rich media that is rewarding to many programmers.

One of the best illustrations of the approach of fun examples for learning programming concepts is the 1993 text *"Oh! Pascal!"* by Cooper and Clancy. Some instructors still remember sparking a student's curiosity and desire to learn through the examples in this book. Today's textbooks seem to be missing this characteristic, as object orient methods have been piled upon control structures, data types, and relational operators leaving little room for fun in a textbook. It is then up to the instructor to develop his/her own motivating examples using graphics libraries.

Options for Graphics in CS1 and CS2

One option for bringing graphics into CS1 and CS2 is to dive straight into MFC. There are many books on MFC programming, but few texts that combine the introduction to programming with MFC. Dietel and Dietel have "Getting Started with Microsoft Visual C++ 6 with an Introduction to MFC." This book is intended as a companion to Dietel and Dietel's popular C++ textbook.

Since MFC's sharp learning curve makes it use in CS1 courses difficult, Cohoon and Davidson have offer a graphics library called EzWindows for use with their book, "C++

Program Design: An Introduction to Progamming and Object-Oriented Design." EzWindows provides a simplistic interface to graphics. It provides limited functionality, but allows the use of graphics in the CS1 course.

This paper describes using the EzWindows library along with MFC programs as an integrated approach to graphics in the beginning programming courses. But there are other options available not discussed in this paper. These include the Carnegie Mellon Graphics libraries, CMUGraphics 1.5 and Carnegie 2.0. The CMU libraries are similar to the EzWindows library in that they provide a simple interface to graphical objects accessible in the CS1 course. Another library is the SOL++2000 Class Library from Antillia.com. This library provides a more sophisticated graphical interface. The interface is similar to MFC in many respects, but may be more approachable to students. Of course there is always Java, but as mentioned earlier, this paper is based on the assumption that C++ will be used to teach the introduction to programming courses.

The following sample programs will illustrate how the EzWindows library and Microsoft Foundation Classes can be used in the CS1 and CS2 courses to illustrate specific course concepts.

Simple Graphics Programming

The EzWindows library is introduced early in the CS1 course with simple programs for drawing geometric shapes in a graphics window. Figure 1 shows a short program that draws a red rectangle on the screen along with a bit map image of a die. It can be used to illustrate objects and how object methods can be called with and without parameters. While there are many other examples of objects that can be used to introduce students to object oriented programming, having visual representations of objects such as rectangles and bitmaps helps some students conceptualize objects better. Having objects tied to graphical representations also suggests to students that object oriented programming is a modern technique worth their attention.

The program also makes use of constructors. The constructors are described in further detail in the Cohoon and Davidson text and can be seen in the header files provided with the library. It is common for students to have trouble understanding constructors when they first encounter them. In EzWindows, constructors are used to associate objects with the windows. It is possible to declare two different windows and construct different shapes in each. This can be used as a concrete example of the need for constructors. EzWindows constructors have a number of other required and optional parameters. One problem with EzWindows and its constructors with required parameters is in declaring vectors of shapes. Having required parameters in the constructor makes declaring and initializing vectors of shapes difficult. Later in the course, though, these constructors can be used as examples of methods with default parameter values and examples of polymorphism.

```
#include "bitmap.h"
#include "rect.h"

// declare a graphics window 5 cm by 10 cm
SimpleWindow W("hello", 5.0, 6.0);

int ApiMain() {
// display the graphics windows
      W.Open();

// draw a rectangle at coordinates 3,4
      RectangleShape R(W, 3,4, Red);
      R.Draw();

// load image from file, draw it
      BitMap Die(W);
      Die.Load("dice3.bmp");
      Die.Draw();

      return 0;
}
```

Figure 1: First EzWindows Program

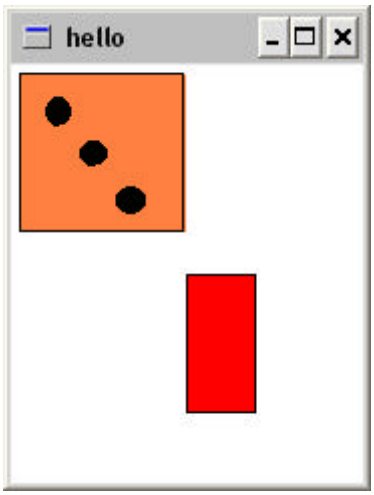

Figure 2: First EzWindows Program Running

The first EzWindows program above also illustrates some concepts common to many GUI environments. This provides a bridge for students from text-based programming to using the MFC library, as well as many other libraries. One such bridge is found in the lack of a true “main()” function in EzWindows programs. EzWindows programs do have an ApiMain() function which is similar to the standard C++ main() function. But this does make students realize that they will not always be writing code in the main() function. This makes the conversion to MFC libraries where no main() functions are visible easier for students.

Another aspect ExWindows programs share with MFC programs is a graphical context. In EzWindows this is represented by the SimpleWindow object. This is analogous to the device context in MFC programs. In EzWindows, all graphics objects will refer to the SimpleWindow object, which is analogous to how all graphics in MFC programs refer to the device context. The program shown in Figure 3 uses the MFC library to illustrate the similarities between the EzWindows program and MFC programs. The MFC program would appear in the CS2 class where the MFC library is introduced. The program is generated using MFC AppWizard, which generated most of the code needed. The program displays a bitmap on the screen, but no code is needed, since this is done during development using the picture control. The program has a button tied to the code shown which draws a rectangle in the window. Adding the code for this button requires the MFC ClassWizard, which generates the empty OnDrawButton method. Students must then add the code shown.

```
// code to run when the Draw Rectangle Button is clicked
void CMISC_mfc1Dlg::OnDrawButton()
{
  // declare a device context for the windows
  CClientDC paintDC(this);

  // declare a brush and select it
  CBrush* red_brush = new CBrush(RGB(255,0,0));
  paintDC.SelectObject(red_brush);

  // Draw the rectangle.
  paintDC.Rectangle(100, 100, 150, 120);

}
```

Figure 3: First MFC Program

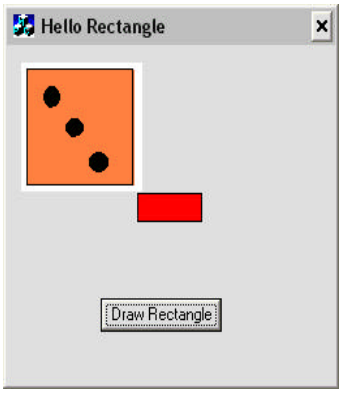

Figure 4: First MFC Program Running

There are some differences between the EzWindows and MFC programs. In EzWindows, the rectangle shape is a separate object, while in MFC it is a method of the

of the display context. MFC requires the creation of a brush before drawing the rectangle and the brush color must be specified using RGB values, while EzWindows provides a set of named colors.

These differences aside, EzWindows provides a nice transition from text program to MFC programming. Students can create graphical programs without having to learn how to use the MFC ClassWizard to assign events to controls. The students also do not have to learn their way through the main files and functions generated by the MFC application wizard.

Event Driven Programming

In our curriculum, students taking CS1 and CS2 have already been introduced to event driven programming. In their freshman year, students are introduced to programming in a three-week portion of our Introduction to Information Systems course—students take CS1 their sophomore year. Currently this introduction to programming is done using Visual Basic and, while the students get a very limited exposure to programming, one concept they do generally come away with is simple event programming. In Visual Basic this is exemplified through mouse click events on command buttons. Students learn to create command buttons on their VB forms and write simple code that executes when those buttons are clicked.

Students in CS1 expect to do similar event driven programming in an environment similar to Visual Basic. The EzWindows and MFC libraries both provide opportunities for this. Handling mouse clicks is one of the few events the EzWindows library provides. This is done by creating a function to act as an event handler and then registering the event handler using the SetMouseClickCallback method. The program shown in Figure 5 shows an example of this. Once an event handler is registered, the student has code that will run whenever the user clicks the mouse within the window. One point to note about the mouse events in EzWindows is that the events are associated with the graphics window and not with the objects in the window. This is different from Visual Basic and MFC where the mouse click events are often associated with buttons or other objects on a window. It is possible to associate mouse clicks with the graphics window in both these environments, and Figure 7 shows an example of an MFC method that handles the mouse clicks in a window.

```
#include <iostream>
using namespace std;
#include "wobject.h"

// declare a graphics window 5 cm by 6 cm
SimpleWindow W("hello", 5.0, 6.0);

// Routine for handling mouse click events
int do_click(const Position &p) {
      // display the position the user clicked in the text window
      cout << "Mouse pressed at " << p.GetXDistance() <<
p.GetYDistance()<< endl;
      return 1;
 }

int ApiMain() {
      // display the graphics windows
      W.Open();
      // Declare what function we handle the mouse click events
      W.SetMouseClickCallback(do_click);

      return 0;
}
```

Figure 5: Event Drive EzWindows Program

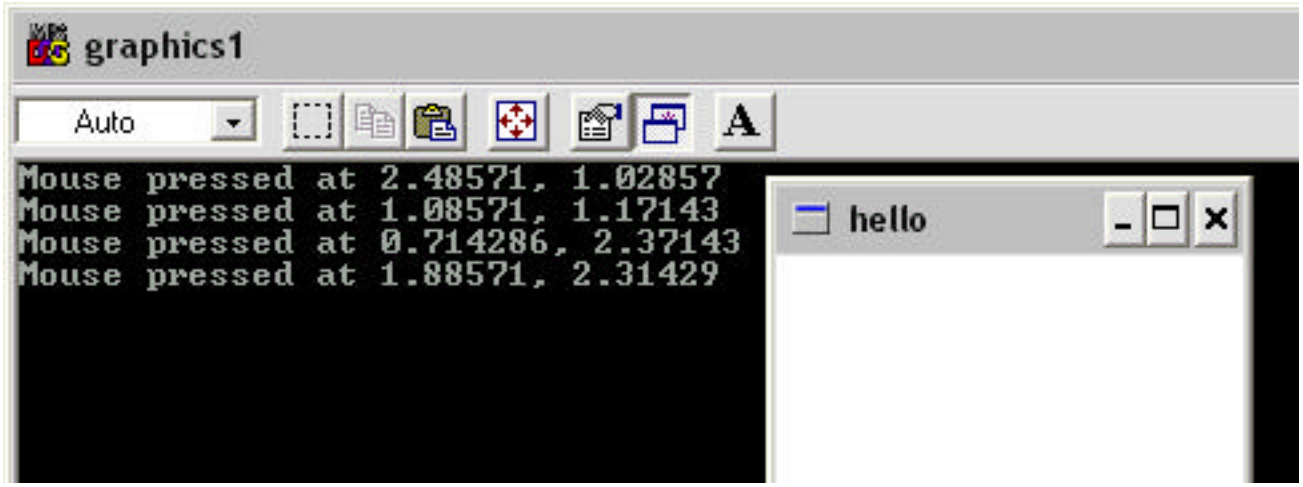

Figure 6: Event Drive EzWindows Program Running

Both the EzWindows example and the MFC example display the coordinates of the mouse click. This introduces the Position object from EzWindows and its analogous object, Cpoint, in the MFC libraries. Both objects store the coordinates for a location and provide methods for accessing these coordinates.

These examples also illustrate a difference between EzWindows and MFC programs. EzWindows programs create a graphics window along with the normal console window. This allows students to continue using standard input and output statements in EzWindows programs, as illustrated by the output stream used in the program shown in Figure 5. Input and output are more difficult in MFC programs, where students must use MFC objects for their I/O needs. This is often not straightforward and can provide a significant roadblock to the use of MFC early in CS1 courses. The example shown in Figure 7 displays output on the window's title bar as one simplified approach.

```
void CMISC_mfc2Dlg::OnLButtonDown(UINT nFlags, CPoint point)
{
      // added by student
      CString message;
      message.Format("Button pressed at %d, %d", point.x, point.y);
      SetWindowText(message);

      // added by class wizard
      CDialog::OnLButtonDown(nFlags, point);
}
```

Figure 7: Event Driven MFC Program

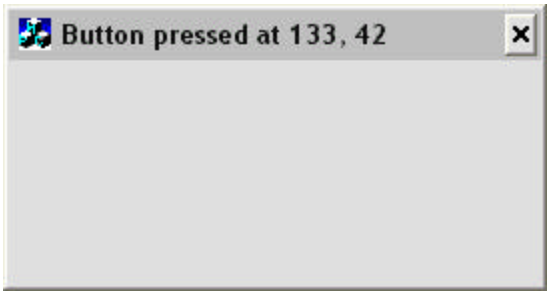

Figure 8: Event Driven MFC Program Running

Conclusion

While the graphical environments for C++ programming are not without flaws, their use can be structured so that students can benefit from their use. Some might argue that precious time in a programming class is lost by learning a GUI library and that this lost time translates into omitted programming concepts. Our experience is that the time spent on learning a GUI library is time spent on some of the more important programming concepts.

References

Cohoon, Janes P. and Davidson, Jack W. (1999), *C++ Program Design: An Introduction to Progamming and Object-Oriented Design*.

Cooper, Doug and Clancy, Michael (1993), *Oh! Pascal! 3rd Ed*.

Deitel, Harvey M. (2000), *Getting Started with Microsoft Visual C++ 6 with an Introduction to MFC*.

Rasala, Richard (2000), Toolkits in First Year Computer Science: A Pedagogical Imperative, *The Proceedings of the Thirty-first SIGCSE Technical Symposium on Computer Science Education.*