USING TERMINAL WINDOW GRAPHICS IN CS1

David Hovemeyer and David Babcock
York College of Pennsylvania
{dhovemey,dbabcock}@ycp.edu

ABSTRACT

Using graphics in introductory programming assignments has been shown to be a useful way to reinforce concepts and engage students. However, the complexity of modern GUI frameworks means that students must be shielded from the underlying complexity of those frameworks, usually through substantial wrapper APIs or project skeletons.

In this paper, we describe *Terminal Window Graphics*—graphics drawn with text characters in a terminal window. This approach allows students to write graphical programs, but is less complex than "real" graphics.

1 INTRODUCTION

As computing power grows, so does the gap in sophistication between programs students use and the programs they can actually implement in an introductory programming course. Some students enter the Computer Science major with the desire to implement sophisticated programs such as 3-D games, and, after writing *n* programs that read from stdin and write to stdout, become disillusioned.

An interesting counter-trend to increasing software complexity is the phenomenon of *retrocomputing*. For example, arcade emulators such as MAME[1] allow classic arcade games to be played on modern operating systems. As a subject matter for programming courses, "retro games" can be useful because they are fun and engaging, simple enough for introductory students to implement, and complex enough to demonstrate interesting algorithms and programming techniques. Previous work [3] exploring the use of retro games as programming assignments has noted a positive student response. For students to complete these assignments, some form of graphics API must be available.

Many previous papers have explored the use of graphics in introductory programming assignments (e.g., [1]). The key prerequisite to making this approach successful is to find a sufficiently small set of abstractions that allow students to write interesting programs, but shield them from the full complexity of the underlying graphics API. Previous work on this problem has focused on developing wrapper APIs around an existing "full-blown" graphics API [4, 5].

Designing a wrapper API is not trivial. An example of a carefully-designed graphics API for introductory programming assignments is the `acm.graphics` API described in the ACM Java Task Force Final Report [4]. This library has been used successfully by many students in many courses. However, it is somewhat complex (18 classes and interfaces), and requires that students program in an object-oriented language. We (the

[1] `http://mamedev.org/`

authors) teach an introductory programming course with a tight schedule, using a non-object-oriented language (C). For these reasons, we were motivated to find the *simplest* possible way to introduce graphical programming.

This paper explores the idea of using a tiny graphics API which, despite its simplicity, allows students to implement interesting graphical programs. We have observed students with no prior knowledge of this API being able to implement interesting animations within a single lab session. The API enables displaying graphics within the *terminal window*.

2 TERMINAL WINDOW PROGRAMMING

All popular operating systems support a terminal application.[2] Each character of output sent to the terminal window is written at the current cursor position advancing the cursor one position to the right, similar to a typewriter. By positioning the cursor at arbitrary locations, images composed of characters can be displayed.

2.1 Terminal Window API

We implemented a small set of API functions to abstract terminal window input and output in an OS-independent manner. These API functions are shown in Figure 1. Although our implementation currently targets C and C++, it could be adapted to other programming languages. Our implementation is available as open source software[3] and can be compiled on Windows, Unix/Linux, and Mac OS X systems.

```
void clear_screen(void);
void move_cursor(int row, int col);
void printw(const char *fmt, ...);
void change_color(int fg, int bg);
void cons_update(void);
int get_screen_height(void);
int get_screen_width(void);
int get_keypress(void);
void sleep_ms(int ms);
```

```
struct Scene s = create_scene();

while (true) {
    render_scene(s);
    cons_update();
    sleep_ms(DELAY);
    s = update_scene(s);
}
```

Figure 1: Terminal Window API functions and example program loop

The `clear_screen()`, `move_cursor()`, and `printw()` functions implement character output to an off-screen buffer. The `change_color()` function changes the current foreground and background colors. The `cons_update()` function copies the contents of the off-screen buffer to the terminal window (providing double buffering.)

Several additional functions are included to support interactive animations. The `get_screen_{width,height}()` functions allow the program to determine the geometry of the terminal window. The `get_keypress()` function does a non-blocking read of the keyboard buffer to determine if a key has been pressed, returning a failure code if the buffer is empty. The `sleep_ms()` function pauses the execution of the program for a specified number of milliseconds.

The most important feature of this API is its simplicity. The entire API consists of 9 functions, with only 5 needed to display images. It has no data types, callback

[2] E.g., the Windows Console, Mac OS X Terminal application, Unix xterm, etc.

[3] `http://libtermgraph.sourceforge.net`

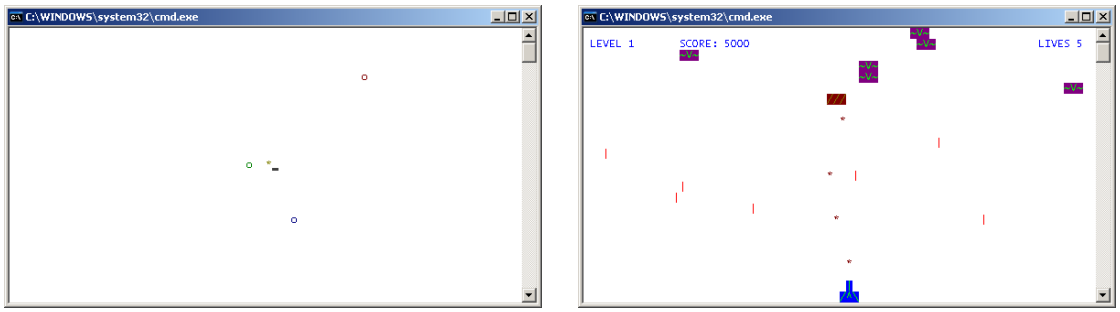

Figure 2: Screenshots of Planet Simulation and Shoot 'Em Up assignments

functions, component hierarchies, etc. The API extends a programming model with which students are familiar (console output using `printf()` and similar functions.)

3 LABS AND ASSIGNMENTS

We have devised a number of lab exercises and programming assignments using the Terminal Window API as part of our introductory programming (CS1) course. Several are described in this section, in order to show both the kinds of visual effects achieved and the programming concepts reinforced.

Each lab and assignment is based on the generalized loop shown in Figure 1 (provided as part of the skeleton code). Students must define the fields of the `Scene` type, and implement the `create_scene()`, `render_scene()`, and `update_scene()` functions. The initial labs and assignments use value semantics for the `Scene` data structure, passing it to functions and returning it from functions by value. Later labs and assignments use pointers to pass the `Scene` by reference.

3.1 Bouncing Character

This lab requires students to animate a single character by adding additional fields to the `Scene` record and code to the `update_scene()` function that allows the character to move in both the x and y directions. The `update_scene()` function is implemented to make the character "bounce" off the boundaries of the terminal window. The lab introduces students to record data types, the mechanics of working with fields, passing records to functions, returning records from functions, and to event-driven programming through the frame-by-frame nature of the animation.

Most students were able to animate a single bouncing character within 30 minutes with one student animating several characters by adding multiple fields to the `Scene` type and additional code in the create/render/update functions.

3.2 Planet Simulation

This assignment builds upon fundamentals presented in the Bouncing Character lab by animating a planet orbiting around a star through simulating the effect of gravity on the planet's direction and velocity and updating the planet's position accordingly. A screenshot of a student's solution is shown in Figure 2. The assignment emphasizes the use of user-defined data types and requires the students to make a distinction between the data model (the `Scene` instance) and the rendering of the model.

Most students were able to complete the basic requirements of the assignment,

with several students choosing to implement additional planets within their simulations.

3.3 Bouncing Particles

This lab exercise extends the Bouncing Character lab by requiring students to create a `Particle` data type in order to model an arbitrary number of objects each with individual positions and velocities along with accessor functions which access the data type through pointers.

Most students were again able to complete the lab within 30 minutes, with several students modifying the window size and increasing the number of particles.

3.4 Shoot 'Em Up

This assignment builds upon the Bouncing Particles lab by implementing an arcade style interactive game similar to the classic game Space Invaders™, where the player moves a ship left and right in response to arrow key presses, and launches multiple missiles at alien spacecraft, destroying them when collisions occur. The assignment requires students to consider the representation of an arbitrary number of different objects, each with independent state and behavior. One suggested approach is to use record data types with arrays and loops to perform initialization, rendering, and control of these objects. A screenshot of a student solution is shown in Figure 2.

Although this was the most challenging assignment of the course, the majority of students successfully completed it, with several students adding extra features such as sound effects, enemy bombs, score keeping, and levels with progressing difficulty.

4 DISCUSSION

Overall, we were pleased with the outcomes of the terminal graphics labs and assignments. In this section, we describe instructor and student experiences, and identify areas for future improvement.

4.1 Instructor Experience

As instructors, we found that once the terminal graphics API was in place, developing new labs and assignments was fairly easy. In general, almost any system that can be modeled and rendered in two dimensions is suitable as a terminal graphics project.

4.2 Student Experience

In this section, we summarize what we learned about the students' experiences completing the terminal graphics labs and assignments.

4.2.1 Anecdotal Experience One of our primary objectives for using terminal graphics in the labs and assignments was to make them *fun*, stimulating student interest and increasing their motivation and sense of accomplishment. As evidence of success, in many assignments students had implemented extra features beyond what was required. For example in the Planet Simulation assignment, a student defined a `Planet` data type to model all 9 planets along with an asteroid belt, each in a different color. In the Shoot 'Em Up assignment, one student implemented sound support using the Windows `PlaySound` API function to add sound effects for firing missiles, hitting enemies, and the explosion of the player's ship. Finally, a student commented to us two weeks after

the due date for the Shoot 'Em Up assignment that he was continuing to work on his program just for fun.

4.2.2 Focus Group Results In order to assess students' opinions about the terminal graphics labs and assignments, we conducted focus group sessions; the key discussion points are summarized below.

- Most students thought that the terminal graphics assignments were interesting and intuitive, but that they were significantly more challenging (sometimes excessively) compared to prior assignments. This was not entirely surprising, given that the terminal graphics assignments occurred towards the end of the semester.
- Many students were able to develop the programs incrementally with a feeling of accomplishment upon completion of each step, and felt that the terminal graphics assignments were conducive to incremental development.
- Several students wished that we had explained how the terminal graphics API functions were implemented[4]. We may address this concern in the future by distributing the API as a pre-compiled library and using the opportunity to discuss the concept of black-box software components.
- Several students also complained about only being required to implement specific functions rather than writing the entire program. In the future, we plan to emphasize the importance of being able to modify and extend existing code.

4.2.3 Survey Results We used the comments gathered from the focus groups to construct an anonymous survey, shown in Figure 3, to solicit opinions on the two terminal graphics assignments along with two prior non-graphics assignments: Bulls & Cows, a guessing game, and Text Adventure, a non-graphical game. The survey was completed by roughly 20 students in the two sections of the course.

Question 1 addressed the ease of use of the terminal graphics API with the majority of students responding positively, supporting our hypothesis that the terminal graphics API is easy for novice programmers to use.

Question 2 verified that the program requirements were understandable, with the two game assignments (Text Adventure and Shoot 'Em Up) slightly more familiar compared to the other two assignments (Bulls & Cows and Planet Simulation).

Question 3 assessed the perceived degree of difficulty for the four assignments, with the trend being that the assignments got progressively more difficult. This result was expected, since the complexity of the assignments increased throughout the semester. However, since 68% of students judged Shoot 'Em Up to be "excessively difficult", we may modify it in the future.

Question 4 assessed the extent to which students found the assignments to be conducive to incremental development, again with mostly positive responses.

Question 5 concerned the amount of intellectual satisfaction the students received from the various assignments, with the student responses clearly showing that the terminal graphics assignments stimulated a greater degree of interest than the non-graphics

[4]Note that no students complained that the API was difficult to *use*.

Question 1: "I found the console graphics functions, such as move_cursor(), printw(), and change_color(), easy to use."

1 = Strongly Disagree, 2 = Disagree, 3 = Neutral,
4 = Agree, 5 = Strongly Agree

1	2	3	4	5
0%	5%	10%	52%	33%

Question 2: "I understood what each program was meant to do."

1 = Very Confusing, 2 = Somewhat Confusing,
3 = Neutral, 4 = Relatively Understandable,
5 = Very Understandable

Assignment	1	2	3	4	5
Bulls & Cows	0%	14%	0%	45%	41%
Text Adventure	0%	0%	9%	18%	73%
Planet Simulation	0%	9%	9%	27%	55%
Shoot 'Em Up	5%	0%	0%	27%	68%

Question 3: "I found the programming assignments to have an appropriate level of difficulty."

1 = Trivial, 2 = Relatively Simple, 3 = Fair,
4 = Challenging, 5 = Excessively Difficult

Assignment	1	2	3	4	5
Bulls & Cows	0%	23%	50%	27%	0%
Text Adventure	0%	18%	41%	36%	5%
Planet Simulation	0%	5%	27%	55%	14%
Shoot 'Em Up	0%	5%	5%	36%	55%

Question 4: "I was able to incrementally develop the program, i.e., add features one at a time."

1 = Strongly Disagree, 2 = Disagree, 3 = Neutral,
4 = Agree, 5 = Strongly Agree

Assignment	1	2	3	4	5
Bulls & Cows	5%	14%	19%	33%	29%
Text Adventure	5%	14%	10%	43%	29%
Planet Simulation	10%	0%	24%	38%	29%
Shoot 'Em Up	10%	10%	33%	10%	38%

Question 5: "I found the programming assignments to be intellectually satisfying."

1 = Very Boring, 2 = Boring, 3 = Neutral,
4 = Interesting, 5 = Very Interesting

Assignment	1	2	3	4	5
Bulls & Cows	5%	14%	38%	38%	5%
Text Adventure	0%	23%	27%	18%	32%
Planet Simulation	0%	18%	23%	32%	27%
Shoot 'Em Up	5%	0%	5%	45%	45%

Question 6: "I feel that working on the assignments improved my programming skills."

1 = No help at all, 2 = Of little help, 3 = Neutral,
4 = Moderately helpful, 5 = Extremely helpful

Assignment	1	2	3	4	5
Bulls & Cows	0%	5%	25%	60%	10%
Text Adventure	0%	10%	29%	24%	38%
Planet Simulation	5%	5%	14%	48%	29%
Shoot 'Em Up	5%	0%	10%	33%	52%

Figure 3: Student survey responses

ones. The Shoot 'Em Up assignment was identified as being particularly satisfying to complete.

Question 6 asked the students to evaluate how much they felt their programming skills were improved by doing the various assignments, with the responses confirming that the terminal graphics assignments, particularly Shoot 'Em Up, were the ones where students felt they learned the most.

Overall, the survey results support our hypothesis that the terminal graphics API provides a valuable programming environment for CS1 students. We do not claim that a graphics-based assignment is *necessarily* more engaging than a non-graphics-based assignment, but rather that assignments using terminal graphics tend to stimulate student interest and are an easy way to introduce graphics in CS1.

5 RELATED WORK

Our work is directly inspired by Stuart Reges's "back to basics" approach to CS1 and CS2 [2]. Many authors have described the advantages of using graphics in introductory programming courses. For example, Roberts describes a simple graphics API for C programs [5], while Astrachan and Rodger describe a series of CS1 projects involving graphics using a custom C++ library [1]. Our work builds on these efforts by attempting to find an extremely lightweight approach to introducing graphics.

REFERENCES

[1] ASTRACHAN, O., AND RODGER, S. H. Animation, visualization, and interaction in CS 1 assignments. In *SIGCSE '98: Proceedings of the twenty-ninth SIGCSE technical symposium on Computer science education* (New York, NY, USA, 1998), ACM, pp. 317–321.

[2] REGES, S. Back to basics in cs1 and cs2. In *SIGCSE '06: Proceedings of the 37th SIGCSE technical symposium on Computer science education* (New York, NY, USA, 2006), ACM, pp. 293–297.

[3] ROBERTS, E. Breakout—nifty assignment. `http://nifty.stanford.edu/2006/roberts-Breakout/`, 2006.

[4] ROBERTS, E., BRUCE, K., CUTLER, R., GRISSOM, J. C. S., KLEE, K., RODGER, S., TREES, F., UTTING, I., AND YELLIN, F. ACM Java Task Force Final Report. `http://jtf.acm.org/rationale/index.html`, 2006.

[5] ROBERTS, E. S. A C-based graphics library for CS1. In *SIGCSE '95: Proceedings of the twenty-sixth SIGCSE technical symposium on Computer science education* (New York, NY, USA, 1995), ACM, pp. 163–167.