QuickDraw: Bringing Graphics into First Year

Ben Stephenson
Department of Computer Science
University of Calgary
2500 University Drive NW
Calgary, Alberta, Canada
ben.stephenson@ucalgary.ca

Craig Taube-Schock
Department of Computer Science
University of Calgary
2500 University Drive NW
Calgary, Alberta, Canada
schock@ucalgary.ca

ABSTRACT

This paper describes a new tool for introducing computer graphics and multimedia applications into first year, called `QuickDraw`, and our experience using it. `QuickDraw` provides an easy to use language and platform independent interface which permits students to create multimedia applications beginning with their first assignment in an introductory computer science course. `QuickDraw` has been carefully designed to avoid complex "magical" statements in order to setup, use, or tear down the multimedia environment, making it an appropriate tool for use with students with no prior programming experience. As instructors, we have found that `QuickDraw` effectively engages students by allowing them to create visually impressive programs with minimal complexity, while continuing to allow us to effectively teach fundamental computer science concepts.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer and Information Science Education—*Computer Science Education*

General Terms

Algorithms, Design, Languages

Keywords

Pedagogy, Computer Science Education, Instructional Tools, Multimedia Projects, Student Engagement

1. INTRODUCTION

Students entering their first university level computer science course today have grown up in an environment where user-level computer use is almost exclusively graphical. Yet, when students enter the university classroom and laboratory, they are often forced into a foreign, predominantly text-based environment which is viewed as cumbersome historical curiosity.

Developing multimedia applications using powerful libraries such as OpenGL or DirectX is far too complex for students entering their first computer science course. As a result, we have developed a tool called `QuickDraw` which reduces the complexity and allows students to easily create multimedia applications. `QuickDraw` was developed with two primary goals:

- **Ease of Use:** The complexity of the underlying graphical and sound environments should be hidden from the students. Students should not need to include any "magical" initialization or shutdown commands.

- **Sustainability:** `QuickDraw` is language and platform independent. As a result, students can continue to use it as they encounter additional languages and platforms over the course of their education.

While there are several other packages available that permit students to create graphical or multimedia applications, these packages are often tied to a specific language and frequently require a collection of "magic" statements in order to initialize and shutdown the system cleanly. They may also require the use of objects, and method invocations on those objects, before these topics have been discussed. In contrast, our system requires only that the students have a reasonable level of understanding of their operating system and the ability to use print statements in whatever language they will be using. Consequently, using `QuickDraw` makes it possible for the first program that a student ever writes to be graphical. Furthermore, our system is both platform and language independent, allowing `QuickDraw` to be used with whatever languages and operating systems are currently in use, or that will be in use in the future. `QuickDraw` is free for Educational and non-commercial use. It can be downloaded from `http://www.cpsc.ucalgary.ca/QuickDraw`.

The remainder of this paper is structured in the following manner. Section 2 describes `QuickDraw`'s features, architecture, and the mechanisms students can use to interact with it. A variety of teaching examples, including several which our students have already implemented as assignments, are discussed in Section 3. Several distinct but related projects are discussed in Section 4. Finally, Section 5 summarizes and presents conclusions.

2. QUICKDRAW ARCHITECTURE

`QuickDraw` was designed to achieve both language and platform independence while also being easy for students with no prior programming experience to use. In order to eliminate ties to a specific language, we developed `Quick-`

Figure 1: **Forming the Connection between a Student's Application and** `QuickDraw`
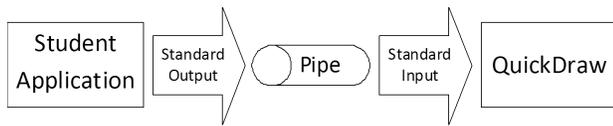
Unix / Mac OSX:

    ./StudentApplication.out | java –jar QuickDraw.jar

Windows:

    StudentApplication.exe | java –jar QuickDraw.jar

`Draw` as an independent application which acts as a rendering engine, performing multimedia operations as requested by the student's program. Data passes from the student's application to `QuickDraw` using pipes, in the same manner as common command line tools such as `grep`. As a result, students can create multimedia applications by using print statements to generate `QuickDraw` commands which are subsequently piped into `QuickDraw`. Since both `QuickDraw` and the student's application interact with a pipe, rather than interacting with each other directly, `QuickDraw` doesn't require any information about the language the student used. The nature of this interaction is depicted graphically in Figure 1.

`QuickDraw` provides platform independence in addition to language independence. Because `QuickDraw` was created using Java, it can be run on any platform where a Java virtual machine is available. Consequently, `QuickDraw` can be used on wide variety of platforms including Windows, MacOS and numerous Unix variants.

`QuickDraw` commands are human readable strings that include a command, followed by a number of parameters. At the present time, `QuickDraw` supports over 100 commands. While this may initially appear to be quite complex, the commands are logically grouped by the behavior they provide, making it possible to ignore commands from subsystems that aren't being used. In practice, impressive 2D graphical results can be displayed using less than a dozen commands. The following subsections provide an overview of the categories of `QuickDraw` commands.

In addition to being easy to use, this architecture also facilitates experimentation. While `QuickDraw` is normally used in conjunction with a pipe, it can also be started without being connect to a student's application. This permits a student to enter `QuickDraw` commands interactively and see the result immediately. We believe that this is particularly useful for gaining familiarity with `QuickDraw`'s features and syntax, as well as gaining experience with graphics concepts such as pixels and mixing colors.

2.1 2D Graphics

`QuickDraw` generates 2D graphical output by providing a variety of graphics primitives such as lines, rectangles, circles, ovals, arcs and polygons. Line colors and styles are set using commands which are separate from the primitives, reducing the number of parameters that must be supplied for each primitive. This design also makes it easy to draw a series of primitives with same color and line style. Fill colors and fill textures are handled in the same manner.
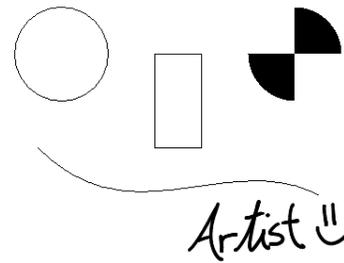

Figure 2: **QuickDraw's 2D Primitives**

```
color 255 255 255
circle 100 100 50
rect 200 100 50 100
cubiccurve 75 200 175 300 275 200 375 250
fillarc 300 50 100 100 0 90
fillarc 300 50 100 100 180 90
loadimage Artist.png artist
drawimage 225 250 artist
```

Figure 3: **Commands used to Generate Figure 2**

In addition, `QuickDraw` also provides support for common image formats such as JPEG and PNG. Images can be loaded and drawn on the screen. Once drawn, images and basic primitives can also be moved or deleted. Grouping is also supported, allowing a set of primitives and/or images to be moved or deleted using a single command.

Figure 2 and Figure 3 show a series of `QuickDraw` commands and the output that those commands generate. A student can generate such output from a program by simply having the program print out the commands, and then pipe the output into `QuickDraw`.

2.2 Turtle Graphics

In addition to the 2D raster graphics primitives described previously, `QuickDraw` also supports 2D vector drawing using turtle graphics. This drawing mechanism has a pen (referred to as a turtle) that can be issued commands such as moving forward or backward and turning left or right. Each movement is normally relative to the turtle's current location, though moving the turtle to a new absolute location is also possible. Turtle graphics are well suited for geometric problems, and have been used previously to introduce children to computer programming [11], as well as teaching more advanced topics such as recursion [6, 9] in a visual manner.

2.3 3D Graphics

`QuickDraw` provides a collection of primitives that can be used to construct a 3D world. Objects can either be placed in world coordinates or camera coordinates, making it possible to have some objects that move with the user as they navigate through the world (such as the front of the car in a driving game). The size of the 3D canvas is also flexible, allowing for status information to be displayed using the two dimensional constructs described previously.

Once the world is constructed it can be navigated with either the keyboard or the mouse using built-in facilities provided by `QuickDraw`. Navigation options include preventing the camera from passing through some objects, and generating input events (described in Section 2.5) when the user comes into contact with an object. Alternatively, programmers can write their own navigation routines by explicitly repositioning the camera in response to user input.

2.4 Audio

Using audio commands allows students to take their applications from being graphical to multimedia. `QuickDraw` supports audio sample playback and a complete text to speech engine capable of speaking arbitrary phrases. Playback is non-blocking, allowing several samples to be played at the same time. Sample playback may also overlap with audio from the text to speech engine.

2.5 Input

`QuickDraw` supports graphical input mechanisms in addition to multimedia output. When requested, events such as key presses and mouse movements can be reported back to the student's application. Higher level input capabilities are also available such as creating basic GUI controls like buttons and list boxes. These controls provide event information such as an item being selected or a button being clicked. Events are also generated by the 3D engine so that a student's application can respond to events such as a collision between the user and an object in the 3D world.

Because information must be passed back to the student's application, use of these controls requires the creation of a bi-directional pipe. Unfortunately this pipe cannot be created using the simple command-line mechanism described previously. Instead, the student must include a call to the appropriate function in their programming language (typically `popen`) to create the bidirectional pipe. While this may initially appear to contradict our desire to eliminate the need for "magic" statements, we believe that by the time we want our students to create applications with the complexity graphical input we will already have discussed the existence of standard libraries and file I/O. As a result, using a call to a library function will be an application of material that students have already learned.

2.6 Getting Help

In addition to extensive external documentation, `QuickDraw` also provides internal facilities that assist students with its use. The simplest of these is the `help` command, which displays a complete list of `QuickDraw`'s commands. Another command name can be provided as an optional parameter to `help`, which will display the usage details for that command.

Students can gain additional help with most `QuickDraw` commands by using the `demo` command. Like `help`, `demo` takes the name of a `QuickDraw` command as its parameter, and then displays information about that command. The information is displayed in two new windows, the first of which contains a sequence of `QuickDraw` commands showing the typical use of the requested command. The second window displays the output generated by those commands. This provides students with concrete examples to work with, including the expected output, which they can modify and extend to suit their purpose.

2.7 Command Summary

This section has outlined the wide variety of commands supported by `QuickDraw`. All of the commands are human readable strings with a consistent structure of a command word followed by zero or more parameters. While many different types of commands exist, simple but visually impressive applications can be generated using only a small subset of them. Using the complete set makes it possible to write extensive multimedia applications.

3. TEACHING EXAMPLES

We have successfully used preliminary versions of `Quick-Draw` in several of our first year courses over the past two years. Our impression is that `QuickDraw` succeeded in engaging students by allowing them to solve interesting graphical problems. The following sections outline a combination of assignments that we have found to be effective and additional assignment ideas that we are considering using in the future.

3.1 First Programs

At our institution, we assume that students come into their first computer science course with no prior programming experience. We simply assume that students have some mathematical background, the ability to reason logically, and an interest in solving problems with a computer. As a result, we must start from the beginning when teaching programming.

Traditionally, the programs that students have written initially are variations on "Hello World", and programs that perform simple arithmetic calculations. These programs provide students with an opportunity to become familiar with the syntax of the programming language, and introduce concepts such as variables, expressions, input and output.

Utilizing `QuickDraw` does not remove the need to start from this introductory level. However, we have found that we can make this introductory material far more interesting by having students create visually interesting programs while learning the same concepts. We observed this first-hand when we used `QuickDraw` for an introductory assignment where students were asked to develop a program to draw a face on the screen. Drawing the face was accomplished by writing a program containing a few print statements, and then piping its output into `QuickDraw`. Several students surprised us by submitting programs that were more than 100 lines long that created visually impressive images. These students clearly went above and beyond what was requested, in a manner that we have never seen previously on a straightforward introductory assignment. We view this as a clear indication that using `QuickDraw` engaged students by giving them the tools necessary to easily generate interesting output – a result that is consistent with a similar introductory image creation project described in [10]. In addition, we have no doubt that those students also benefited from their extra effort by gaining greater familiarity with the syntax of the language.

This simple assignment idea can be extended to include practice using variables, expressions and input by having the user enter two values which specify where the image should be located on the screen. To accomplish this goal, students must write expressions that compute how far each graphics primitive is from the specified location while also allowing them to explore outputting a mixture of variables and literals using a single print statement.

3.2 Loops

With `QuickDraw`, it is possible to present numerous interesting examples which demonstrate looping concepts. One assignment we have found particularly effective has students graph a polynomial function. In the simplest case, the function is graphed in screen coordinates as a set of points. Additional experience working with expressions can be added by changing the location of the origin on the screen or scaling

the function so that a change of 1 pixel does not correspond to a change of 1.0 in the x or y value of the function. Additional loop complexity can be introduced by requiring students to graph multiple functions or mark the minimum and maximum points on a graph of a higher degree polynomial function.

3.3 Arrays

Two dimensional arrays are a natural representation for images. Using nested loops, the images stored in a two dimensional array can be manipulated in many interesting ways. With `QuickDraw`, it is also easy to display the image, both before and after the transformation is performed.

We have successfully used `QuickDraw` to teach students about arrays by having them implement image transformations such as rotation, pixelation, and chroma keying. Several additional image transformations that are appropriate for an introductory course are described in [2, 3] and [16], including blurring the image, computing an intensity histogram and edge detection.

3.4 Recursion

Recursion is often introduced to students using the calculation of n factorial and Fibonacci numbers as motivating examples. Unfortunately, when computing n factorial recursively, students are often left wondering why they wouldn't simply use a loop. Fibonacci numbers are also problematic because computing them recursively is prohibitively slow. As instructors, we make claims about the utility of recursion, but often fail to provide compelling uses for it without introducing complex data structures such as trees.

Last year, we used `QuickDraw` and two graphical problems to motivate the study of recursion. The first problem we considered was the creation of fractal images. Several examples of fractal images were examined, and students were asked to implement functions for drawing the Sierpinski Triangle and Fractal T-Square. Several additional fractal images that are appropriate for use in an introductory course have been documented in [7].

The second example we considered was how to implement the paint bucket tool available in most drawing programs. This tool is used to change the color of an arbitrarily shaped monochromatic region in an image, without changing anything outside of that region. We found this example to be particularly appropriate because it brought recursion together with the student's previous work involving two dimensional arrays in order to implement a tool that many students have used in the past.

While each of these problems can be completed iteratively, the iterative solutions are far from obvious to most students in an introductory course, and even if discovered, are more cumbersome than the recursive solution. As a result, we believe that exploring these graphical problems using `Quick-Draw` conveyed the utility and value of recursion to students much more effectively than traditional examples.

3.5 Graphs

Textual representations of graphs and trees are difficult to interpret. By displaying these structures graphically, students can gain increased appreciation for the relationships between the nodes. The task of displaying graphs also provides students with an appreciation for the challenges associated with creating effective data visualization tools.

We have used assignments involving graphs with first year students to model food chains within an ecosystem and to simulate the propagation of a virus within a population. For the food chain assignment, students were provided with a series of text files, each describing an ecosystem in terms of Species m eats Species n. Students were required to identify a suitable encoding for the ecosystem (in the form of a directed graph) and to construct these structures in memory by parsing each of the provided text files. Once the ecosystem was encoded, students had to compute its key components: key species, top species, species dependencies and the names of all species within the ecosystem. `QuickDraw` was employed to help the students understand the complexity contained within each ecosystem. Specifically, when students were asked to draw each ecosystem, it became readily apparent to them that the complexity of a graph is not only affected by the number of nodes within the graph, but is also affected by the number of relationships between nodes. Graphical display and visualization were key to developing this understanding.

For the viral propagation assignment, students were provided with text files which described communities. These communities contained persons (nodes) and relationships between persons (edges) through which viral propagation could occur. The simulation was given two main parameters for propagation control:

1. The number of days before the person was removed from the simulation (due to death or quarantine).

2. The virulence of the virus expressed as a probability of transmission (1.0 = guaranteed propagation).

Students were required to write a program which would simulate the propagation of viruses through given populations. They were also required to develop hypotheses regarding the factors which affect viral propagation, and to design experiments (carried out using their simulator) to test these hypotheses. As the students progressed through their experiments, they quickly realized that there is a third parameter which affects viral propagation which is the topology of the population's graph. This insight was only discovered because the students were required to visualize the population using `QuickDraw`. As a series of input files, the overall topology of the system is masked; once the data can be visualized, the structure of the system and the ability to observe its effects becomes clear.

3.6 Object Oriented Design

Previous work [5] has described using a raytracer as a compelling graphical example for teaching object oriented analysis and design. While creating the raytracer, students practiced their object oriented design skills by creating classes to represent shapes in the scene, vectors, rays and light sources. The output from the ray tracer was a graphics file in a common format.

This project is well suited for use with `QuickDraw`. In addition to abstracting the details of the graphics library, using `QuickDraw` would allow students to see the image as it is rendered. We believe that one of the greatest strengths of this project is its ability to engage students with a compelling, visual, object oriented problem, and that providing immediate feedback would enhance the level of engagement.

We have used also used a nuclear reactor simulation visualized with `QuickDraw` to teach students about object oriented design. The purpose of the simulation was to help

the students understand how an object model analysis of a complex system can be abstracted using classes which can be further abstracted using inheritance. Where students typically have difficulty with object-oriented programming is that they tend to ignore the object model once it has been abstracted into a class model. Using `QuickDraw` to provide a graphical representation of the nuclear reactor simulation forced the students to continue to focus on the object model because it is the object model (the reactor components) which was displayed. By maintaining the student's focus on the object, class and generalized class models, we believe that the students ultimately developed a much stronger understanding of object-oriented programming than if a simple text-based output mechanism were used.

3.7 Examples Summary

This section has outlined numerous sample problems that demonstrate and reinforce core introductory computer science topics in a visually interesting manner. We have successfully used many of these examples over the past two years with a prototype version of `QuickDraw`, and we look forward to using several of these problems to engage students in the coming year.

4. RELATED WORK

Declining enrollment in computer science programs across North America has emphasized the need to actively work to attract, engage and retain computer science students [1]. This has led to a number of initiatives to introduce graphical problems into introductory computer science courses. These initiatives include programming environments such as Alice [4] and numerous high level libraries [8, 12, 13, 14, 15].

`QuickDraw` is similar to each of these projects, providing student engagement through the creation of visually interesting applications. However, `QuickDraw` is also distinct from all of these projects because it completely separates the graphics library from the student's program. This separation provides benefits that include complete language and platform independence, and a high level abstraction that hides all of the complexity of the underlying graphics, input and audio libraries.

By making `QuickDraw` language and platform independent, students can continue to use `QuickDraw` to write graphical applications as they advance in their program and work with additional languages and computing environments. Because the protocol for interacting with `QuickDraw` remains unchanged, previous experience is directly applicable to subsequent courses. This allows students can concentrate on learning course concepts rather than the details of a new graphics library. In addition, this separation also provides flexibility in the program curriculum, allowing the languages or computing environments for a specific course to be updated without requiring instructors and teaching assistants to learn a new graphical environment.

5. CONCLUSION

This paper has described `QuickDraw`, a tool that allows first year students to write graphical applications, and our experience using it. `QuickDraw` is easy to use, and provides both language and platform independence. In our experience, `QuickDraw` is an effective mechanism for providing interesting, engaging examples and assignments without burdening students with the complexity normally associated with graphical applications.

6. REFERENCES

[1] Sanjeev Arora and Bernard Chazelle. Is the thrill gone? *Commun. ACM*, 48(8):31–33, 2005.

[2] Kevin R. Burger. Teaching two-dimensional array concepts in Java with image processing examples. *SIGCSE Bull.*, 35(1):205–209, 2003.

[3] Jeffrey W. Chastine and Jon A. Preston. Teaching 2d arrays using real-time video filters. In *SIGITE '05: Proceedings of the 6th conference on Information technology education*, pages 133–137, New York, NY, USA, 2005. ACM.

[4] Stephen Cooper, Wanda Dann, and Randy Pausch. Alice: a 3-d tool for introductory programming concepts. In *CCSC '00: Proceedings of the fifth annual CCSC northeastern conference on The journal of computing in small colleges*, pages 107–116, , USA, 2000. Consortium for Computing Sciences in Colleges.

[5] T. A. Davis. Graphics-based learning in first-year computer science. *Computer Graphics Forum*, 26(4):737–742, December 2007.

[6] Yehoshafat Shafee Give'on. Teaching recursive programming using parallel multi-turtle graphics. *Comput. Educ.*, 16(3):267–280, 1991.

[7] Aaron Gordon. Teaching recursion using recursively-generated geometric designs. *J. Comput. Small Coll.*, 22(1):124–130, 2006.

[8] R. Jiménez-Peris, S. Khuri, and M. Patino-Martínez. Adding breadth to CS1 and CS2 courses through visual and interactive programming projects. In *SIGCSE '99: The proceedings of the thirtieth SIGCSE technical symposium on Computer science education*, pages 252–256, New York, NY, USA, 1999. ACM.

[9] Ivan B. Liss and Thomas C. McMillan. Fractals with turtle graphics: a CS2 programming exercise for introducing recursion. In *SIGCSE '87: Proceedings of the eighteenth SIGCSE technical symposium on Computer science education*, pages 141–147, New York, NY, USA, 1987. ACM.

[10] Sarah Matzko and Timothy A. Davis. Teaching CS1 with graphics and c. *SIGCSE Bull.*, 38(3):168–172, 2006.

[11] Seymour Papert. *Mindstorms: children, computers, and powerful ideas.* Basic Books, Inc., New York, NY, USA, 1980.

[12] Eric Roberts and Antoine Picard. Designing a Java graphics library for CS 1. *SIGCSE Bull.*, 30(3):213–218, 1998.

[13] Eric S. Roberts. A c-based graphics library for CS1. *SIGCSE Bull.*, 27(1):163–167, 1995.

[14] Gerald Shultz. Integrating 3d graphics into early CS courses. *J. Comput. Small Coll.*, 21(3):169–178, 2006.

[15] Kelvin Sung, Michael Panitz, Scott Wallace, Ruth Anderson, and John Nordlinger. Game-themed programming assignments: the faculty perspective. *SIGCSE Bull.*, 40(1):300–304, 2008.

[16] Richard Wicentowski and Tia Newhall. Using image processing projects to teach CS1 topics. *SIGCSE Bull.*, 37(1):287–291, 2005.