USING AN OPEN-SOURCE LANGUAGE TO TEACH INTRODUCTORY PROGRAMMING AND GRAPHICS CONCEPTS IN FIRST-YEAR ENGINEERING AND TECHNOLOGY COURSES

Magesh Chandramouli and Ge Jin, Purdue University Calumet; Patrick E. Connolly, Purdue University

Abstract

In this paper, the authors propose the use of an open-source programming environment called Processing© in first -year undergraduate engineering and technology education courses. Currently, there is significant use of proprietary commercial software in technology education. The use of such tools for teaching introductory graphics principles to students has resulted in a heavy dependence on commercial software, which in turn has financial implications for the educational institution. On the other hand, open-source programming languages (PL) reduce the financial burden for both institutions and students. Processing© is an open-source PL that can be used to teach basic programming and graphics concepts to first-year engineering/technology students. The syntax of Processing© is relatively straightforward. Students can develop intermediate-level graphic applications in a relatively shorter time period with less complexity. Also, Processing© can serve as a bridge to facilitate the subsequent transition to more advanced Object-Oriented Programming (OOP) languages like Java and C-Sharp as Processing© itself is built on OO principles. In this paper, the authors explain how to use Processing© to design and implement freshman-level graphics courses in the disciplines of engineering and technology. The results presented show that students are receptive to the idea of learning the concepts of computer graphics using a programming medium. Overall, the results evince that the visual- and graphics-based lecture samples and lab exercises helped students learn complex concepts.

Introduction

The computer graphics technology program at Purdue University Calumet (PUC) does not include many programming courses at the freshman and sophomore levels. To better prepare the students to meet industry requirements and to enhance their employability by sharpening their programming skills, it is essential to teach introductory graphics programming course(s) during the first and second years. The undergraduate Computer Graphics Technology (CGT) program at PUC focuses on areas such as multimedia design, web design and development, computer animation, game development and graphic design. The plan of study of the CGT program consists of six courses in computer animation and game development, seven courses in multimedia and web design, five courses in graphic design, five foundational courses in sketch, raster and vector visualization, and one graphics programming course.

Formerly, CGT students would take programming courses (Java or C++) from other academic departments, but these often would be too discipline-specific to meet the needs of the CGT students. To resolve this issue, a CGT-specific PL course was developed. Since CGT students tend to be visually oriented, and are exposed to various graphic design and animation tools (e.g., Photoshop©, Illustrator©, 3D Max©, Maya©, etc.), it was anticipated that the development of a course of this type would be beneficial. It was also believed that the course would serve the dual purpose of enhancing general programming skills and inculcating computer graphics concepts. The programming course is required for all CGT undergraduate students and meets the general education requirement of all undergraduate students in STEM disciplines at PUC.

Several key components impacting student motivation must be considered when discussing programming instruction and learning. Driscoll [1] stated that instructional material must appeal to learners and must also motivate learners in their goal achievement. Keller and Litchfield [2] claimed that considering student motivation "is particularly important because it pertains to a person's basic decisions as to whether or not to accept responsibility for a task and to pursue a given goal". As applied to instruction, Talton and Fitzpatrick [3] noted: "A long-standing difficulty in the development of introductory courses in computer graphics is balancing the educational necessity of ensuring mastery of fundamental graphics concepts with the highly desirable goal of exciting and inspiring students to further study by enabling them to produce visually interesting programming projects."

Many researchers in education have shown that the use of instructional methods other than the traditional lecture format is much more effective in facilitating student learning. Methods such as active learning [4], problem solving [5], [6], and project-based learning [7], [8] are encouraged as ways of exciting students, providing real-world problem-solving experiences, and increasing transfer of critical skill-sets from the classroom to the workplace, especially in introductory programming instruction [9] and other technology-based learning [6], [10], [11]. To allow students to relate new learning to existing skills/knowledge without cognitive overload, teaching in technology environments should include as much contextual content as possible [1].

Teaching a foundational programming course to freshman is a challenging task. The difficulty of passing a programming course at the freshman level has resulted in high drop-out rates in computer science (CS) and information technology (IT) majors in higher education worldwide [12]. Computing educators stated that learning programming languages and acquiring problem-solving skills are time-consuming and difficult tasks [13], [14]. One important reason is that the current programming languages are mainly designed for industrial use and, hence, are way too complicated for teaching programming foundations. Jenkins [15] indicated that the programming language for freshman should be chosen for educational purposes rather than its popularity in industry. These points further corroborate the authors' rationale for choosing Processing© as a medium for teaching foundational programming and graphics skills to freshmen.

A freshman programming course is an important foundational course in computer science and graphics education. Educators in these disciplines have proposed various ways to enhance the learning of programming concepts in freshman courses. Recently, Hernandez et al. [16] taught fundamental programming principles to freshmen students through 2D Game Engine: GameMaker. Game Engine allows students to understand fundamental programming principles without having to learn the complex syntax of programming languages. Similarly, Kazimoglu [17] felt that digital game play could be an effective method for computational thinking programming instruction. Additionally, Papastergiou [18] showed that a digital gaming method of learning was both more motivational and more successful than traditional non-gaming methods in teaching computer memory basics to high school students. However, Holzinger et al. [19] cautioned that although dynamic media can be an effective learning tool, excessive use of such methods can lead to cognitive overload if not utilized judiciously. Another example along these lines would be the failure of PASCAL as a PL tool in CS and graphics education. PASCAL is a strongly typed procedural PL that was designed mainly for computer science education. Unfortunately, PASCAL has evolved into a complex industrial-level PL and, therefore, lost its merit in CS education. To overcome this complexity, several new programming languages were introduced [20]. Processing© is one such language which was introduced especially for visual artists, making it less programming-intensive [21]. From the above discussion, the choice of Processing© for the freshman course discussed in this study is evident.

Teaching Programming Fundamentals

Programming skills have become crucial for engineering and technology students irrespective of their major discipline. Programming tasks inculcate problem-solving and critical-thinking skills in students. Typically, upon graduation, students may enter industry or pursue graduate studies. Problem-solving and critical-thinking are essential skills for succeeding in both industry and academia. Furthermore, immediately after the freshman year, students continue to take several Engineering Technology (ET) courses that involve various assignments and projects. Students can accomplish several important components of these projects using programming tools or can even complete an entirely programming-based project that meets the course requirements.

In Processing©, a programming statement to print a sentence to the console would be just a single line such as: println('hi! Good day!'). On the other hand, printing a small phrase to the console would still require several lines of code. This will be explained further with examples in the Results and Discussion section. That which can be accomplished with Processing© in a single straight-forward statement takes several lines of code. This by no means implies that Java is not as efficient as Processing©. Of course, Processing© is built on Java and employs the basic notions on which Java is based. However, the argument here is that for students at the beginner level, learning programming via Processing© can be much easier than using a language like Java. Java keywords in the above program such as 'public', 'static', 'void', etc. are based on concepts such as variable scope, return-type, etc., which may be too complex for a student at the beginner level.

Besides enabling students to learn programming skills, Processing© teaches students a very important notion called object-oriented programming (OOP). Extremely successful PLs like Java and C++ are based on the notion of OOP. Processing© can be considered a stepping stone for Java.

Knowledge of Processing© helps students upgrade to learning Java at a later stage. The ultimate objective of using Processing© is to facilitate the learning of introductory programming and graphics concepts by first-year students. When teaching programming fundamentals to students, 'cognitive overload' is a key issue [1], [19]. This can be caused by several factors, including material complexity, failure to integrate new material with previous knowledge and disruptive elements that distract from learning the materials. Courses loaded with perplexing jargon or complex programming terminology can easily discourage students at the beginner level. Also, it gets tiring when students need to do a lot of coding before they can see the actual result, which is the case with languages like C++, OpenGL, Java, etc. Instead of directly learning programming using languages such as Java or C++, if a simpler language such as Processing© is used, the learning process becomes a smooth one. With Processing©, students can create simple shapes and programs using very little code, which reduces the gap between learning and executing.

Some programming notions are generic and span different languages. These include: data, variables, scope, functions (or methods/routines) loops, iteration, conditionals, recursion, etc. Illustrations can quickly communicate programming concepts, especially for students with visual learning style preferences [22-24]. For instance, one of the fundamental programming concepts is loops. Graphics examples may help students to transcend the abstract level and take the learning process to a more concrete level. This is because, when using Processing©, students can actually see the visual result of using loops via the graphic output window. In Processing©, this graphic output is called a "sketch". This iterative logic is used to illustrate the three main loops: for, while and do-while (see Figure 1). This shows a circle repeated diagonally, a face-like object, and a checkerboard representation created using loops. From left to right, Figures 1a-1c represent a sequential learning process, going from simple to complex graphics objects: iterative do… while loop for circle primitives; while loop to create faces from primitives; nested for loop to create an array or grid. Another important concept in programming is conditional expressions. Upon satisfying a condition, a particular set of commands are executed, else, another set of commands are executed. One exercise involves generating a random float value less than 200. The program's function was to draw a black/yellow checkerboard pattern if the value < 100 or a blue/red pattern if the value generated was > 100 (see Figure 2). The visual nature of graphics plays a dual role of aiding the learning process and reinforcing material learned.

Programming Elements

Despite the plethora of PLs available today, all programming languages are built upon some fundamental concepts such as data types, variables, constants, functions, iterations, and so on. 'Brevity' is a salient advantage when using Processing© for teaching the introductory programming terms. In other words, when using Processing© [25], the above terms and associated concepts can be explained using very few lines of code. Creating a simple program to store data and print it out can involve several lines of code in other

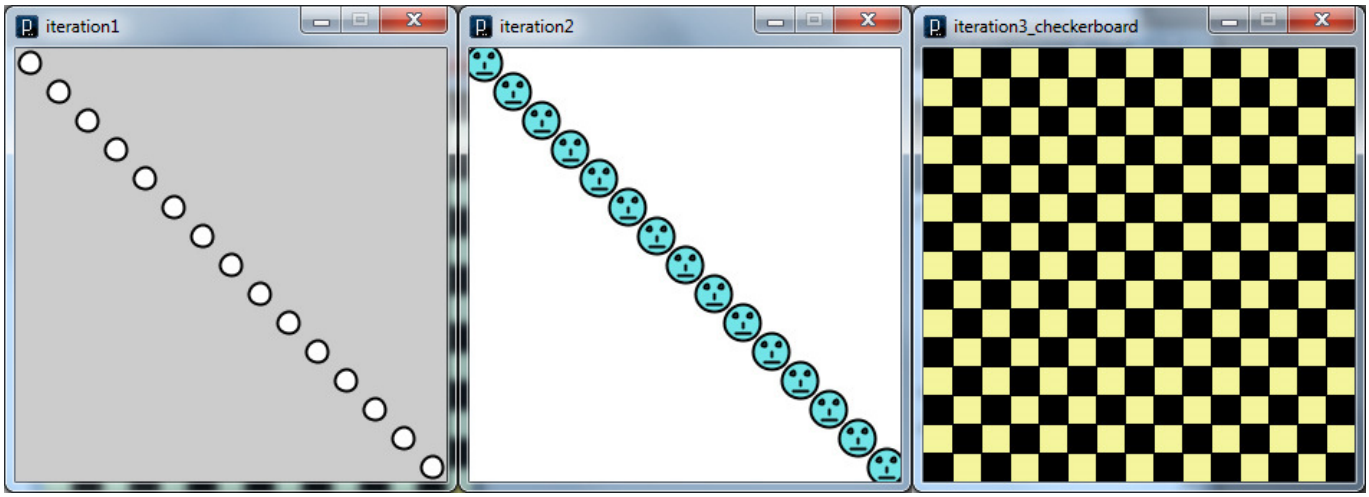

Figure 1. Illustration of Loops Using Simple Object Geometries
(a) Circular Objects ('do…while')
(b) Face-like objects ('while' loop)
(c) Checkerboard ('for' loop)

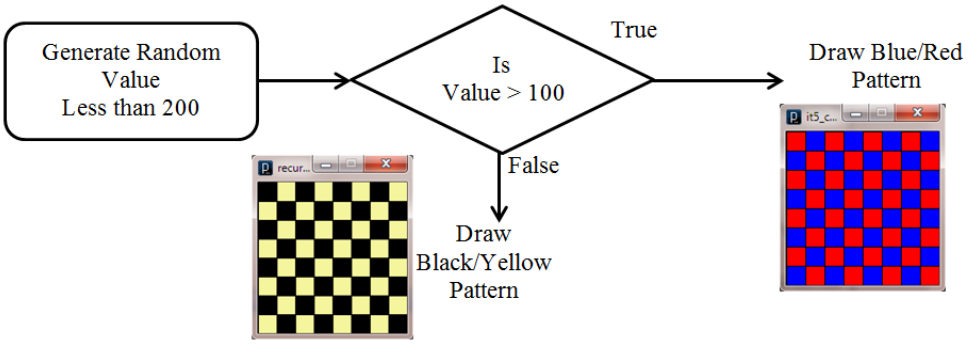

Figure 2. Conditional Expression 'if..else'

Abstract

PLs. On the other hand, 2 or 3 lines of simple Processing© code can clearly explain basic programming concepts.

Data and Declaration

Simply stated, a variable is a data-holder or a container used to hold data. Just as in the real-world, different containers are used to hold different types of materials, and different variables are used to store different types of data. Let us consider the very simple code snippet, int x = 25; this line of code creates a variable, x, of type integer and assigns it a value of 25. Data is useless unless it can be applied. With Processing©, this can be done instantaneously. This value stored in x can be used by adding a line of code—line (0,0,x,x);—thereby generating a line from the origin (0,0) to (25,25). Concepts covered by these two brief lines of code include:

- Data-type (integer)
- Variable declaration (int x)
- Assignment (x = 25)
- Using variable (line(0,0,x,x))
- Parameters (enclosed between parentheses)

This brief example illustrates the simplicity as well as immense potential of Processing©. In several other programming languages, explaining and understanding the terms and concepts mentioned here could have involved numerous lines of code, which was accomplished in only two small programming statements.

Functions

Functions are also known as method/routines in different programming languages. Functions are used to facilitate code reusability and readability, while reducing redundancy. Typically, functions can be classified into two broad categories: built-in and user-defined. The former refers to pre-existing functions provided to the user by the programming language. Common examples include print(), println(), etc. The second type of function, known as user-defined function (UDF), is the primary means by which a programmer writes a customized program for a specific application. Here, the process involves two steps: teaching the principle and using the function in graphics. The first step is to teach the fundamental concept of the function. A simple example of calculating the sum of two numbers is used. This function computes the sum of integers x and y and returns the value. This example shows input parameters and return type in a function.

```
int sum(int x, int y)
{ return x+y; }
```

The next step is extending the previous example to illustrate the use of a function in spatial transformation (translation) in computer graphics. The translate function takes the input of the original 2D point and then translates it by the value of T. (return value = translated coordinate)

```
Point2D translate(Point2D x, Point2D T)
{ return x+T; }
```

Graphics Fundamentals

Similarly, on the graphics side, various concepts are common across all graphics programming languages such as points, lines, polygons, primitives, compound shapes, transformations, etc. The use of a high-level programming language necessitates that students learn and complete many basic language-specific coding exercises before creating their first program or graphics object. However, the straightforward commands in Processing© such as point() and line() enable students to create simple shapes instantaneously.

Students can create simple shapes and programs using very little code. This reduces the gap between learning and executing. To create a circle, a single line of command is sufficient; ellipse(50,50,50,50) will create a circle with a radius of 50 pixels in the center of the default Processing© window. A simple 'face' can be created by putting together primitive shapes, as shown in Figure 3.

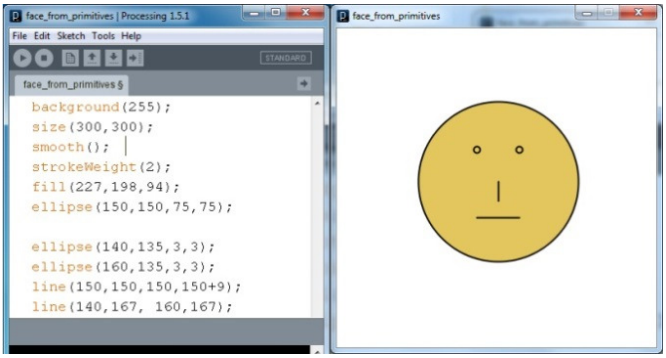

Figure 3. Simple Object using Primitives

The basic graphics primitives such as points, lines and polygons can be explained to students using straightforward single line code:

1. Point(50, 50) creates a point at 2D coordinate (50,50)
2. Line(0, 0, 50, 50) creates a line from 2D coordinate (0,0) to (50,50)
3. Rect(50, 50, 50, 50) creates a rectangular polygon at the corner (50,50) with height/width=50

Transformations constitute an important element of any graphic modeling environment. Instead of using complex 3D objects to explain the notion of transformations, simple primitives can be used to efficiently communicate the concept of transformation. Considering the flowing diagram, all shapes used are essentially circles. The scene consists of circles that have been transformed (scaled and translated). Similar to the prior scene, which used only circles, just by using primitives and applying rotation/translation, a simple Clock can be created. (see Figure 4).

In the following sections, these graphics fundamentals are explained, in particular two extremely important programming paradigms: Object Oriented Programming and Event Driven Programming.

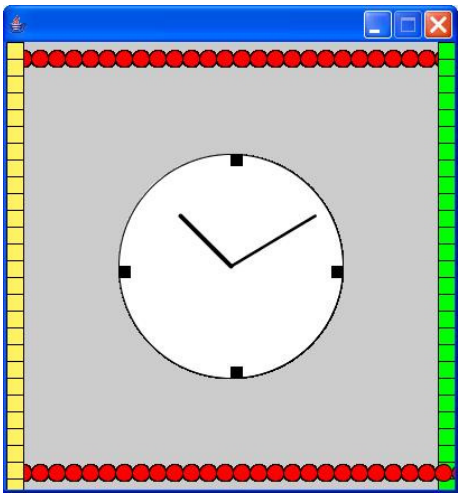

Figure 4. Clock Object from Classes (OOP)

Objects and Object Oriented Programming

The concept of Object Oriented Programming (OOP) is relatively difficult for freshman college students to understand. Therefore, in this course, only the following essential topics of OOP were covered: Class/Object, Data/Method, Private/Public keyword and inheritance. To describe the concept of Class and Object, illustrations of various cars such as SUV, Convertible, Coupe, Hatchback, Sedan, etc. were used. Each individual car is an object, and all cars share some common characteristics. All of the cars can be described using a common 'car' class. Extending this further to explain inheritance, geometric shapes were used. A Shape class is a parent class for Circle and Rectangle classes, and the Rectangle class is a parent class for a Square class. Also, as an object is made of several other objects, the different objects can be grouped into one coordinate system or into different coordinate systems in order to achieve desired movements. Additional OOP topics covered in the class were used to create an object using default and non-default constructors. In the following example, a Clock object is created using objects belonging to the Circle class. Loops were used to create the border designs. Continuing to the next paradigm mentioned earlier, the same clock was animated based on the event-driven programming notion, as explained below.

Events and Event-Driven Programming

An 'event' is the critical feature in the event-based or event-driven programming paradigm. An event acts as the trigger that initiates an action. A wide range of possible occurrences can serve as triggers or events and a wide range of responses can occur. A computer, washing machine, electric oven or television can be made to respond to events appropriately. This is because they all contain something (data) and, upon being initiated (or put to use), they perform specific tasks. In other words, they have their own properties and they can 'act' or behave. An object may be acted upon and it reacts in accordance with the action. The mouse and keyboard event and interaction topics are introduced using graphical drawing applications. For the mouse interaction, the mouse button pressed, released and mouse move events were covered.

The mouse and keyboard events are ubiquitous in most modern programming languages. However, the lack of visual elements may make the teaching of mouse interaction concepts a very tedious process because students may have difficulty in relating to such new logic. Nevertheless, the use of visual media significantly reduces the cognitive load. Most students in the CGT program have used Photoshop and Illustrator to create 2D graphic images. In these graphics programs, the mouse and keyboard are commonly used to draw custom shapes and move the created shapes. To illustrate mouse click, mouse move and mouse release events, Bezier curve and object movement applications were generated (see Figure 5). The Bezier curve drawing application mimics the Pen tool in Photoshop and Illustrator. The left mouse click will create a curve vertex on the 2D screen. The user can drag the mouse in any direction in order to define the tangent vector of the curve. If the user releases the mouse button immediately after mouse click, a corner vertex will be created. The mouse pressed, mouse released and mouse move events were combined to implement curve drawing application. The object movement application requires the user to click on one of the pre-drawn shapes and move it to a new location. When the mouse button is pressed, the program will check if the mouse cursor is inside an object. If it is inside, the object will be selected, and the user can move the object to a new location with the mouse move event.

The clock created in the earlier lab exercise (Figure 4) is animated so that the hands of the clock show the actual (current) time. Using loops, the hands of the Clock object are animated. In Processing©, there is a function called draw () that draws objects to the graphic output window at a specific rate. Typically, this is 60fps (frames per second). Using this repetitive drawing of objects to the screen and by using a counter variable to track the number of iterations, the new positions of the hands (hour hand, minute hand, and second hand) are incremented and drawn continually to the screen. This produces the final outcome of an animated clock. This builds upon the notion of integrating objects generated from classes with the concept of event-driven programming to produce an animated version of the clock object shown earlier.

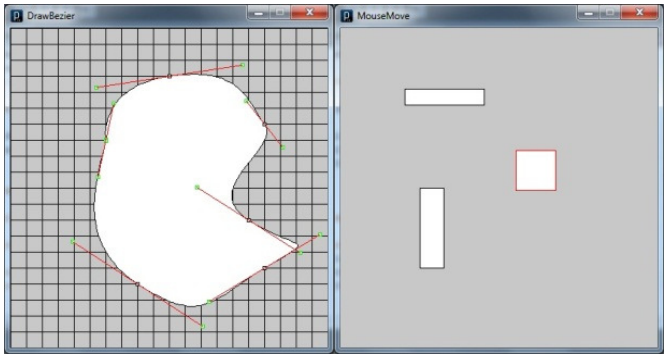

Figure 5. Curve Drawing and Object Manipulation using Mouse Event and Interaction

The lack of such visual elements may make understanding programming concepts very tedious. Hence, a visual medium was employed to reduce the cognitive load and facilitate understanding.

Results and Discussion

This course was designed to cover essential programming and graphics concepts in an easy-to-understand manner. The exams/quizzes and labs/projects were also designed based on the aforementioned points. Table 1 shows the grading criteria.

Table 1. Grading Criteria

Activity	%	Scale	Grade
Exercises	20%	90-100	A
Project	20%	80-89	B
Participation	10%	70-79	C
Midterm Exam	25%	60-69	D
Final Exam	25%	<60	F
Total	100%		

Table 2 shows some of the sample exam/quiz questions. The exams and quizzes included different types of questions that reinforce theory, actual programming code and graphics elements administered in the form of MCQ (Multiple Choice Questions), fill in the blank, and True or False questions administered via the Purdue Blackboard system. Being an introductory course, the level of difficulty of the exam questions and lab/project requirements was maintained between simple and intermediate. Students were also provided with the Processing© software. This course covered two kinds of materials:

1. Introductory material for freshmen in the Computer Graphics Technology program that introduces them to general university life [26].
2. An introduction to Computer Graphics using Processing© (a Java Variant).

The exercises included materials from the introductory freshman material and the Processing-based CG component. Students also completed a Processing-based project. These projects included generating scenes with simple and compound graphic objects. A compound object is made from multiple Processing© objects. Students created a wide range of scenes on a 900-by-900 Processing© canvas (sketch) using graphic tools such as color, background, fill, stroke weight, etc. that resulted in creative graphic designs.

The midterm and final exams consisted of 100 questions that tested students' understanding of graphics programming concepts. Table 3 illustrates the number of questions in each sub-category, and the students' performance in the corresponding sub-category. The exam results indicate that students performed extremely well in understanding 2D graphics transformation concepts. The results also showed that approximately 80% of the students correctly answered basic programming questions. These questions covered concepts such as variable, conditional statement, loop, array, OOP and event-driven programming. Overall, the results evinced that the visual- and graphics-based lecture samples and lab exercises helped the students learn complex concepts. The function part had the lowest rate (72%). An analysis of the questions in that category showed that the low rate was due to the recursive concept. While this approach worked well in teaching basic programming concepts, limitations exist in teaching 'recursion'.

In summary, a comprehensive table showing all of the advantages of using this Open Source language is given in Table 4. The table encapsulates the advantages of using Processing© for a freshman-year course. Processing© also proved to be beneficial from the perspective of software procurement. This is a significant advantage when compared to commercial software packages that impose significant financial hardship on the university and the students. Relying heavily on expensive software packages for teaching computer graphics courses can contribute to increasing tuition costs. As Processing© is based on Java language, students will find transitioning to higher-level programming including Java / C++ and other Object Oriented courses

Table 2. Sample Exam Questions Based On Processing©

	Question Type	Question	Answer Choices***

Sample Qn. 1.	MCQ* (Programming/ Graphics)	What will be output displayed in the console?
		int x = 125;
		float y ;
		y = float(x);
		println(y);
			1. 125
			2. 12.5
			3. 1.25
			4. 125.0
			5. None of the above

Sample Qn. 2.	MCQ / Theory	For Bezier curve, the middle parameters are *Control Points* which define the shape of the curve.	*True* False
Sample Qn. 3.	TF** (Theory/Prog)	A Processing© program can have any number of draw() functions	True *False*

Sample Qn. 4.	MCQ* (Programming/ Graphics)	This program generates how many circles?
		size(500,500);
		int x = 1;
		while(x > 100)
		{ellipse(150+x,250+x,150,150); };
			1. 5
			2. 20
			3. 9
			4. 14
			5. None of the above

*MCQ - Multiple Choice Question **TF – True or False *** For the reader's convenience, the correct answer is underlined here.			

Table 3. Question Distribution and Student Success Rates in Midterm/Final Exam

	Midterm (50 Questions) and Final (50 Questions)	
	Number of Question	Success Rate
Variable	10	79.6%
Conditional Statement	12	86.6%
Loop	6	89.2%
Array	9	78.9%
Functions	14	72.2%
Graphics Transformation	9	95.1%
Object Oriented Programming	14	84.5%
User Interaction(Keyboard/Mouse)	12	89.8%
Images/Text	5	76.7%
Processing© Specific Functions	9	89.5 %
Total	100	83.9%

smoother and easier. Two types of end-of-semester course evaluations were conducted: Course Learning Outcomes and Instructor Evaluation. The Course Learning Outcomes are presented in Table 5. In order to further confirm the positive results demonstrated in this section, the following question was included in the final instructor evaluation: 'I am glad that I took a Graphics Programming course in my freshman year'. Fifty percent of the students replied 'Strongly Agree' and 50% selected 'Agree'. This confirms that students are receptive to the idea of learning CG concepts using a programming medium.

Conclusion

In this paper, the authors propose using an Open Source programming environment called Processing© in first-year undergraduate engineering and technology education courses. The study explored the reduction of the gap between learning and executing by using Processing©, which is a Java-based language that is easily understood by first-year students in engineering and technology. With Processing©, students can create simple shapes and programs using a small amount of code, which reduces the gap between learning and executing. The study also addressed the issue of reducing cognitive overload by minimizing perplexing jargon or complex programming terminology. Even though Processing© was used as a medium for teaching introductory graphics concepts, the use of a graphic software package also had another notable advantage. Using graphics illustrations served as an effective means to communicate programming notions.

When using Processing©, the students could actually see the visual results of using loops via the graphic output window. This facilitated understanding important programming concepts including loops/iteration, recursion, functions/ methods, etc. In addition, Processing© serves as a link facilitating the subsequent transition to more advanced programming and Object-Oriented Programming (OOP) languages like Java and C-Sharp as Processing© itself is built on OOP principles. In summary, then, the authors explain the use of Processing© to design and implement freshman-level graphics courses in the disciplines of engineering and technology. This can be extended and modified further in the forthcoming years to lay a foundation for junior- and senior-year coursework involving EXL (Experiential Learning) projects.

References

[1] Driscoll, M. P. (2005). *Psychology of Learning for Instruction*. (3rd ed.). Needham Heights, MA: Allyn and Bacon.

[2] Keller, J. M., & Litchfield, B. C. (2002). Motivation and Performance. *Trends and Issues in Instructional Design and Technology*. (R. A. Reiser & J. V. Dempsey, Eds.). Upper Saddle River, NJ: Pearson Education.

[3] Talton, J. O., & Fitzpatrick, D. (2007). Teaching Graphics with the Opengl Shading Language. *ACM SIGCSE Bulletin, 39*(1), 259-263. doi: 10.114.

Table 4. Advantages of Processing© for Teaching Graphics/Programming to Beginners

	Processing©	Other Popular OO languages like Java/C++
Syntax	Common tasks like printing and drawing simple graphic primitives can be performed using simple, brief programming statements	Getting familiarized with the syntax involves time and practice. Even simple tasks like printing to the console or creating a sample graphic object involves writing several lines of code
API (Application Programming Interface)	Very simple, easy-to-understand graphics programming interface, especially for beginners	Understanding the Java API is a time-consuming process and involves considerable time input
Graphics Programming	Users can create graphic objects straightaway using simple one-line statements	Typically, a good understanding of the basic programming concepts of Java/C++ is required before starting graphic programming
Object-Oriented Programming	OO notions such as Class, Methods, Inheritance can be demonstrated using few lines of code and simple graphic examples	Employing OO using Java/C++ entails understanding class and instance variables, class and instance methods, inheritance etc. , which may be challenging for a beginner-level programmer
Event-Driven Programming	Mouse and Keyboard events can be handled by calling mousePressed() and keyPressed() functions. The mouse location can be directly acquired from pre-defined variables.	To enable the mouse interaction in the Java, a programmer needs to import MouseEvent and MouseListener package, inherit MouseListener class to the frame window, and overwrite mouse event handling methods. Mouse position is acquired by calling a MouseEvent object's getX() / getY().
Installation	Straightforward (The software can be downloaded in a single zip file. Software can immediately be used after unzipping it.)	Installation procedure might involve multiple steps and often times, post-installation procedures may be involved (e.g. setting environment variables in Java, etc.)
Importing Libraries	User only needs to copy library files to a predetermined processing folder. Then the library can be easily imported from Processing© menu.	In C++, user needs to specify header file, library file and DLL files using complicated user interface. In source code the corresponding header file should be included by the user. In Java, the precompiled Jar file should be added to the project and the user needs to the import correct package into the Java source file.
Execution	No such explicit conversion needs to be performed by the user	In C++/Java, program first needs to be compiled into byte code files before execution.

Table 5. Question Distribution and Student Success Rates in Midterm/Final Exam

Survey Question (1-Lowest, 5-Highest)	Score
1.I learned basic programming concepts such as variables, data types, selection and repetition control structures, arrays, files, and methods and functions from this course.	4.375
2. After taking this course, I can create simple object-oriented applications.	4.19
3. After this course, I can understand the similarities between Processing© and Java.	3.875
4. After this course, I can apply knowledge to different computer graphics software applications.	4.3125

[4] Prince, M. J., & Felder, R. M. (2006). Inductive Teaching and Learning Methods: Definitions, Comparisons, and Research Bases. *Journal of Engineering Education, 95*(2), 123-138.

[5] Jonassen, D. H. (2002). Integration of Problem Solving into Instructional Design. (R. A. Reiser & J. V. Dempsey, Eds.). *Trends and Issues in Instructional Design and Technology*. Upper Saddle River, NJ: Pearson Education.

[6] Newby, T. J., Stepich, D. A., Lehman, J. D., & Russell, J. D., & Leftwich, A. T. (2010). *Educational Technology for Teaching and Learning*. (4th ed.). Upper Saddle River, NJ: Pearson Education.

[7] Hadim, H. A., & Esche, S. K. (2002). Enhancing the Engineering Curriculum through Project-based Learning. *Proceedings of the 32nd ASEE/IEEE Frontiers in Education Conference.* Boston.

[8] Mills, J. E., & Treagust, D. F. (2003). Engineering Education – Is Problem-based or Project-based Learning the Answer? *Australasian Journal of Engineering Education*, 2-16. Retrieved from http://www.aaee.com.au/journal/2003/mills_treagust03.pdf.

[9] Pears, A., Seidman, S., Malmi, L., Mannila, L., Adams, E., Bennedsen, J., et al. (2007). A Survey of

Literature on the Teaching on Introductory Programming. *ACM SIGCSE Bulletin, 39*(4), 204-223. doi: 10.1145/1345375.1345441.

[10] Jonassen, D. H., Howland, J, Moore, J., & Marra, R. M. (2003). *Learning to Solve Problems with Technology: A Constructivist Perspective*. (2nd ed.). Upper Saddle River, NJ: Pearson Education.

[11] Roblyer, M. D. (2004). *Integrating Educational Technology into Teaching*. (3rd ed.). Upper Saddle River, NJ: Pearson Education.

[12] Kinnunen, P. and Malmi, L. (2006). Why Students Drop Out CS1 Course? *Proceedings of the 2006 International Workshop on Computing Education Research*, 97-108.

[13] Lahtinen, E., Ala-Mutak, K., & Jarvinen, H. (2005). A Study of the Difficulties of Novice Programmers. *ACMSIGCSE Bulletin, 37*(3), 14-18.

[14] Gomes, A., & Mendes, A. J. (2007) Learning to Program - Difficulties and Solutions. *Proceedings of the International Conference on Engineering Education (ICEE 2007)*, 283-287.

[15] Jenkins, T. (2002). On the Difficulty of Learning to Program. *Proceedings of the 3rd Annual conference of the LTSN Centre for Information and Computer Sciences*, 53-38.

[16] Hernandez, C. C., Silva, L., Segura, R. A., Schimiguel, J., Paradela Ledón, M. F., Bezerra, L. M., et al. (2010). Teaching Programming Principles through a Game Engine. *CLEI ELECTRONIC JOURNAL, 13*(2), 3.

[17] Kazimoglu, C., Kiernan, M., Bacon, L., & MacKinnon, L. (2012). Learning Programming at the Computational Thinking Level via Digital Game Play. *Procedia Computer Science 9,* 522-531.

[18] Papastergiou, M. (2009). Digital Game-based Learning in High School Computer Science Education: Impact on Educational Effectiveness and Student Motivation. *Computers & Education 52,* 1-12.

[19] Holzinger, A., Kickmeier-Rust, M., & Albert D. (2008). Dynamic Media in Computer Science Education; Content Complexity and Learning Performance: Is Less More? *Educational Technology & Society, 11* (1), 279-290.

[20] Sinapova, L. (2005). Teaching "Principles of Programming Languages" through Design and Implementation of a Simple Programming Language. *Proceedings of the Midwest Instruction and Computing Symposium 2005 (MICS 2005)*. Paper 104.

[21] Reas, C., & Fry, B. (2006). Processing: Programming for Media Arts. *AI & Society, 20*(4), 526-538.

[22] Thomas, L., Ratcliffe, M., Woodbury, J., & Jarman, E. (2002). Learning Styles and Performance in the Introductory Programming Sequence. *ACMSIGCSE Bulletin, 34*(1), 33-37.

[23] Lewalter, D. (2003). Cognitive Strategies for Learning from Static and Dynamic Visuals. *Learning and Instruction 13,* 177-189.

[24] Zualkernan, I. A., Allert, J., & Qadah, G. Z. (2006). Learning Styles of Computer Programming Students: A Middle Eastern and American comparison. *IEEE Transactions on Education, 49*(4), 443-450.

[25] Reas, C., & Fry, B. (2003). Processing: a learning environment for creating interactive web graphics. *Proceedings of the SIGGRAPH 2003 Conference on Web Graphics*, SESSION: Java applications. San Diego.

[26] Brickman, E., Thinnes, D., & Osmon, B. (2009). *Freshman Year Experience: Plan for Success,* (3rd Ed.). Kendall Hunt Publishing Company.

Biographies

MAGESH CHANDRAMOULI is an Assistant Professor of Computer Graphics Technology at Purdue University, Calumet. He earned his Bachelor of Engineering from the College of Engineering, Anna University, India and M.Eng from the Faculty of Engineering, National University of Singapore. He earned his Master of Science from the Scuhlich School of Engineering, University of Calgary, Canada and PhD from Purdue University, USA. His research interests include computer graphics, 3D Visualization, Multi-objective Optimization, Genetic Algorithms, Virtual Reality, and Computer Animation. Dr. Chandramouli may be reached at mchandr@purduecal.edu

GE JIN is an Assistant Professor of Computer Information Technology and Graphics at Purdue University, Calumet. He earned his B.S. degree (Computer Science, 1997) from Peking University, China, MS (Computer Science, 2000) from Seoul National University, Korea, and Doctor of Science degree (Computer Science, 2007) from the George Washington University, DC. Dr. Jin is currently teaching at the Purdue University Calumet. His research interests are in computer graphics, virtual reality, computer animation, medical visualization, and educational game development. Dr. Jin may be reached at ge.jin@purduecal.edu

PATRICK CONNOLLY is a Professor of Computer Graphics Technology at Purdue University. He earned his B.S. degree (Design and Graphics Technology, 1982) and MS (Computer Integrated Manufacturing, 1983) from Brigham Young University, and Ph.D. (Educational Technology, 2007) from Purdue University. Dr. Connolly's research interests are in visualization, spatial ability development, virtual and augmented reality, and product data and lifecycle management. Dr. Connolly may be reached at connollp@purdue.edu