Visualization and Motivation in Introductory Programming Courses

Literature review

Cameron Adams
Department of Computer Science
University of Cape Town
Cape Town, South Africa
ADMCAM003@myuct.ac.za

ABSTRACT

Computational skills are becoming more prevalent in related and previously unrelated fields. The teaching and learning of introductory programming courses has been reported to be difficult. New ways of motivating and engaging students are being researched and developed.

This paper reviews what visualization tools have been used to engage, motivate and teach introductory programming, and concludes by identifying a gap in the literature - namely the lack of empirical experiments using graphics libraries to further research and how creative expression can promote motivation.

CCS CONCEPTS

• **Computer Science education** → **Visual tools**; • **Visualization** → *Graphics libraries*;

KEYWORDS

Graphics libraries, visualization, literature review

1 INTRODUCTION

The world is becoming immensely more digital and the computer has massively contributed to the growth of civilization. A global acknowledgement of the importance of Computer Science education is growing. [8] Previously unrelated professions, such as journalism and law, are starting to need technical programming skills. [2]

Computational thinking (CT) is becoming more widespread. Many people, who have no particular interest in Computer Science, but who engage in social media, typically discuss the results of the underlying recommendation systems. Computer Science enrollment at Universities is growing fast and the challenge still remains on how to capitalise on the increased interest and to expand the enrollment among women and minorities. [9]

Learning to program is not easy. [5] Neatly summarised the problems into a) teaching dynamic concepts that use static materials, b) learning programming is practical and some students tend to just read textbooks, c) students need adequate mathematical and logical thinking, d) complex syntax and a demand for high level abstract thinking can be challenging, e) there is an image of what a typical programmer is are some people are put off.

A proposed tool which allows students to 3D print a 3D artefact which they create, using graphics libraries, causes for this literature review, which gives a brief overview of CS education research, how visualization aids teaching and learning introductory programming, and identify gaps in the literature - namely where creative expression can engage and motivate new students.

2 COMPUTER SCIENCE EDUCATION

Literature in the teaching of introductory programming can be categorized into four sub-fields: curricula, pedagogy, language choice and tools for teaching. [6]

Curricula: Countries generally have their own highly influential groups to discuss and plan new curricula. In the US, they have major professional computing societies (the ACM and IEEE Computer Society) and in Europe they have the Bologna Accord. Every introductory computing curricula has a programming metaphor and paradigm. Metaphors like "computation as calculation" or "computation as interaction". Paradigm examples are object oriented or functional programming.

Pedagogy: While the curriculum defines the material to be taught, pedagogy deals with the manner in which the material is taught, learnt and managed in order to get desired learning outcomes.

Language choices: Many languages have been used for teaching introductory programming. Factors such as faculty preference, industry relevance, complexity and syntax influence the choice.

Tools for teaching: Tool research papers were divided into three sub-fields. Visualization tools, automated assessment tools and programming environments.

Automated assessment tools: These are needed for the high enrollment. They help teachers check the correctness of a program's execution and provide a way for students to learn from errors and improve work before final submission. A limit to the number of submissions can mitigate the case where students might exploit a trial-and-error approach to programming.

Programming environments: Integrated Development Environment (IDE) increases productivity by organising program components, highlighting syntax, and visualizing debuggers.

Visualization tools are particularly relevant to this paper so in the next section, these tools will be discussed, as well as a motivation for visual learning.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

3 VISUALIZATION

Visual learning has lagged behind relative to verbal learning [4]. Visual representations attract and maintain motivation. Visualization provides an additional way of representing information and encourages the learning that students may not get from text alone. Visual literacy is a crucial component of technological literacy and its development is one of the main goals of STEM education. [16]

Program visualization (PV) has been defined by many authors but the general idea is: the use of various techniques to enhance the human understanding of computer programs. Examples are debuggers, IDEs and, something we will discuss in detail in section 4, graphics libraries. These examples visualize *implemented* code. With this definition, physical computing, which is the use of microcontrollers to control electronics, such as LEDs and switches, could be considered as program visualizations. [14]

Visual programming (VP) is the use of "visual" techniques to specify a program. Examples are Scratch and Alice which are environments that do not necessarily use code but instead use drag-and-drop or other visual abstractions to interface with 'read' code to then implement a sequence of computational steps.

Algorithm visualization (AV) is the visualization of a high-level description of a piece of software. AVs are a subset of PVs, as a program may include multiple different algorithms.

Software visualization (SV) can be defined to include all of these as a means to reduce ambiguity and covers all of the software design process from planning to implementation. [11]

These SV tools aid lectures to teach and students to learn though there are different levels of engaging with a SV, that have been shown are more effective than others. A 2002 study by Hundhausen et al. [10] argued that no matter how well crafted a visualization is, it is of little value unless students are in an active learning activity. The study's purpose was to develop a taxonomy to provide a framework for conducting empirical experiments that attempt to evaluate the effectiveness of AV in teaching. The six categories in the taxonomy are: 1) No viewing, 2) Viewing, 3) Responding, 4) Changing, 5) Constructing, 6) Presenting

Grisson, McNally and Naps [7] in 2004 investigated the influence of visualizing *sorting* algorithms in lectures on students' learning using this taxonomy. The study compared 3 different ways students could engage the sorting algorithm visualization. Engagement levels were (1) not showing, (2) showing and (3) allowing interaction. Results showed that students that were allowed higher levels of engagement in the visualization scored higher marks.

4 GRAPHICS LIBRARIES

4.1 Creating visual artefacts

Drawing APIs / graphics libraries and some programming languages were created to produce graphics to visualize data, to get better insight, and to create aesthetic visuals to promote motivation and engagement in novice programmers, and in recent years, as a creative outlet.

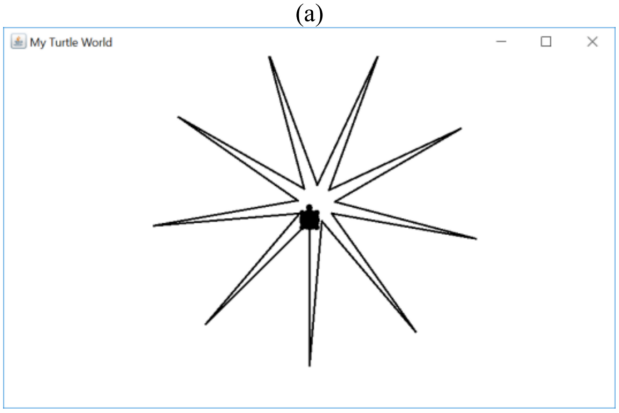
Some of these tools were created as early as 1967. The earliest programming language to do so is Logo, created in 1967 by Seymour Papert to teach children the basics of programming [15]. The turtle in Logo is a pointer object that can be moved around based on simple instructions. Lines are drawn following the turtle's movement.

4.2 Use of graphics libraries in CS education

A recent 2020 paper by Bakar, Mukhtar and Khalid [1] pointed out that teaching and learning programming is not easy. They used TurtleGraphics to aid a 14 week course teaching the fundamental concepts of programming. A few of these concepts were repetition structure, conditional structure, arrays and array processing. After the course, students' motivation levels were measured via the ARCS model (attention, relevance, confidence, and satisfaction) by allowing them to rank their level of agreement from 1 to 5 to statements such as "I enjoy learning TurtleGraphics in the programming course." (attention), "TurtleGraphics approach helps me to understand a basic programming concept." (relevance) and "Completing a programming assignment using TurtleGraphics gives me satisfaction." (satisfaction). The results showed that students' motivation from the four aspects of the ARCS model were high, with the overall motivation value of 4.2870 (out of the maximum score of 5). This shows that this approach has potential in teaching introductory programming.

```
for(int i = 1; i <= 18; i++) {
      if (i % 2 == 0)
            turtle.turnRight(175);
      else
            turtle.turnRight(225);
      turtle.forward(150);
}
```

(a)



(b)

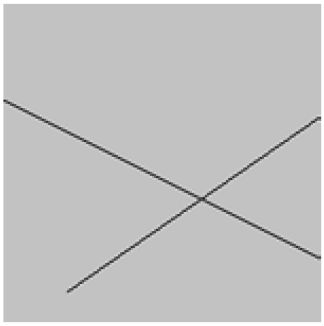
Figure 1: Use of TurtleGraphics in teaching repetition structure [1]. (a) produces (b) after a World canvas is instantiated

The use of graphics libraries to promote engagement and motivation in introductory courses have been reported decades ago. A 2001 paper by Becker [3] found that teaching introductory Java from Textbooks does not introduce object orientated concepts early and thorough enough as students need time to master these concepts. Becker discovered Karel the Robot, an educational graphics library, created in 1981, that can control a simple robot named Karel that lives in an environment consisting of a grid of streets (left-right) and avenues (up-down). Karel understands five basic instructions: move (move to next square he is facing), turnLeft turns 90 degrees left), putBeeper (places a beeper/dot in the square he is standing in), pickBeeper (lifts the beeper in the square he is standing in) and

turnOff (switches off and ends the program). Becker reported that students enjoyed directing robots to do various tasks.

A more recent graphics library, called Processing, was created by Ben Fry and Casey Reas in 2001. It was created to serve as a software sketchbook and to teach programming fundamentals within a visual context [12]. Its design was to "feel" like a scripting language while providing the same capabilities as a more complete language like Java. The syntax to get started and produce a visual on a canvas is minimal which caters for novice programmers yet slightly more competent programmers can use what Java skills they have to produce complex visuals.

No empirical evidence on the use of Processing in introductory programming course has been reported [13] though it has been said to used in computing curricula in U.S. universities.



```
size(200,200);
line(40, 180, 200, 70);
line(0, 60, 200, 160);
```

Figure 2: Minimalist syntax to produce a visual in Processing [13]. First line creates a 200x200 pixel canvas and the last two lines draw two black lines on the canvas

5 CONCLUSION

Overall, it is evident from the literature that the teaching and learning introductory programming is a challenge and thus further research in new ways of motivating students is needed in the future. In conclusion, there are some gaps in the current state of programming visualization. There appears to be a lack of empirical experiments in using graphics libraries in the teaching of introductory programming. The use of a 3D printer and an easy to syntax graphics library can engage students at the highest levels of the engagement taxonomy [10]; which are Constructing and Presenting which as shown in study [7] produce the highest learning. Being creative with code and producing physical artefacts opens a new conversation between mathematical logically inclined students, and more expressive creatively inclined students which can. This has potential to increase, in particular, Confidence and Satisfaction in the ARCS model for motivation, as students will be able to socialize and present their work.

REFERENCES

[1] Marini Abu Bakar, Muriati Mukhtar, and Fariza Khalid. 2020. The Effect of Turtle Graphics Approach on Students' Motivation to Learn Programming A Case Study in a Malaysian University. *International Journal of Information and Education Technology* 10, 4 (2020).

[2] Evan Barba and Stevie Chancellor. 2015. Tangible Media Approaches to Introductory Computer Science. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '15)*. Association for Computing Machinery, New York, NY, USA, 207–212. https://doi.org/10.1145/2729094.2742612

[3] Byron Weber Becker. 2001. Teaching CS1 with Karel the Robot in Java. In *Proceedings of the Thirty-Second SIGCSE Technical Symposium on Computer Science Education (SIGCSE '01)*. Association for Computing Machinery, New York, NY, USA, 50–54. https://doi.org/10.1145/364447.364536

[4] Michelle Patrick Cook. 2006. Visual representations in science education: The influence of prior knowledge and cognitive load theory on instructional design principles. *Science education* 90, 6 (2006), 1073–1091.

[5] Anabela Gomes and António José Mendes. 2007. Learning to program-difficulties and solutions. In *International Conference on Engineering Education–ICEE*, Vol. 2007.

[6] Scott Grissom, Myles F. McNally, and Tom Naps. 2003. Algorithm Visualization in CS Education: Comparing Levels of Student Engagement. In *Proceedings of the 2003 ACM Symposium on Software Visualization (SoftVis '03)*. Association for Computing Machinery, New York, NY, USA, 87–94. https://doi.org/10.1145/774833.774846

[7] Scott Grissom, Myles F. McNally, and Tom Naps. 2003. Algorithm Visualization in CS Education: Comparing Levels of Student Engagement. In *Proceedings of the 2003 ACM Symposium on Software Visualization (SoftVis '03)*. Association for Computing Machinery, New York, NY, USA, 87–94. https://doi.org/10.1145/774833.774846

[8] S. Hodges, S. Sentance, J. Finney, and T. Ball. 2020. Physical Computing: A Key Element of Modern Computer Science Education. *Computer* 53, 4 (2020), 20–30.

[9] Adams Nager and Robert D Atkinson. 2016. The case for improving US computer science education. *Available at SSRN 3066335* (2016).

[10] Thomas L. Naps, Guido Rößling, Vicki Almstrum, Wanda Dann, Rudolf Fleischer, Chris Hundhausen, Ari Korhonen, Lauri Malmi, Myles McNally, Susan Rodger, and J. Ángel Velázquez-Iturbide. 2002. Exploring the Role of Visualization and Engagement in Computer Science Education. *SIGCSE Bull.* 35, 2 (June 2002), 131–152. https://doi.org/10.1145/782941.782998

[11] Blaine A Price, Ronald M Baecker, and Ian S Small. 1993. A principled taxonomy of software visualization. *Journal of Visual Languages & Computing* 4, 3 (1993), 211–266.

[12] Casey. Reas. *Processing : a programming handbook for visual designers and artists*. MIT Press, Cambridge, Mass.

[13] Casey Reas and Ben Fry. 2006. Processing: Programming for the Media Arts. *AI Soc.* 20, 4 (Aug. 2006), 526–538. https://doi.org/10.1007/s00146-006-0050-9

[14] Robert H. Seidman. 2009. Alice First: 3D Interactive Game Programming. In *Proceedings of the 14th Annual ACM SIGCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE '09)*. Association for Computing Machinery, New York, NY, USA, 345. https://doi.org/10.1145/1562877.1562986

[15] Cynthia Solomon, Brian HARVEY, Ken Kahn, Henry Lieberman, Mark L Miller, Margaret Minsky, Artemis Papert, and Brian Silverman. 2020. History of Logo. (2020).

[16] Igor Verner and Amir Merksamer. 2015. Digital Design and 3D Printing in Technology Teacher Education. *Procedia CIRP* 36 (2015), 182 – 186. https://doi.org/10.1016/j.procir.2015.08.041 CIRP 25th Design Conference Innovative Product Creation.