Creating Explanatory Visualizations of Algorithms for Active Learning

Jonathan C. Roberts* Bangor University, UK
James Jackson† Bangor University, UK
Christopher Headleand‡ University of Lincoln, UK
Panagiotis D. Ritsos§ University of Chester, UK

Abstract

Visualizations have been used to explain algorithms to learners, in order to help them understand complex processes. These 'explanatory visualizations' can help learners understand computer algorithms and data-structures. But most are created by an educator and merely watched by the learner. In this paper, we explain how we get learners to plan and develop their own explanatory visualizations of algorithms. By actively developing their own visualizations learners gain a deeper insight of the algorithms that they are explaining. These depictions can also help other learners understand the algorithm.

Keywords: Explanatory visualization, Information Visualization, Teaching visualization, Learning Support

Index Terms: H.5.2 [Interfaces and Presentation]: User Interfaces—Graphical User interfaces (GUI). K.3.2 [Computing Milieux]: Computers & Education—Computer Science Education

1 Introduction

Learners often find computer graphics and scientific visualization algorithms complex and difficult to understand. *Explanatory visualizations* can help tell the story of how a process occurs or an algorithm works on a dataset. For instance, displaying the different cases of the Marching Cubes algorithm [4] is demonstrated more easily by animating the cases with a test dataset and an explanatory visualization presentation.

Computing education is an area where explanatory visualizations have been successfully used. For example, searching the internet for "visualization of sorting algorithms" produces a large number of informative examples. Most of the search results are animation videos and while they do explain the algorithms, they have been hand-crafted by the educator and are merely 'viewed' by the learner. There are some examples where the learner can interact with a tool to change parameters and tryout different algorithm configurations, but still the workload – and especially the design and creative imagination – resides with the educator.

We propose a different strategy, where the learner is actively involved in making their own explanatory visualization. This *experiential learning* approach enables the learner to engage with the design and makes their own decisions as to how the data is visualized, taking ownership of their learning. While this active learning process may take longer it gives the learner a much deeper understanding of the underlying principles of the algorithms. Students comprehend better the important steps and can explain the concepts in a discerning way.

Our methodology is to get students to individually think, plan, and develop their own explanatory visualizations. The students critique their work throughout the process and so develop their critical thinking, in addition to their application skills. Fig. 1 shows our Explanatory Visualization active learning cycle, broken down into six stages. We demonstrate the work, with the case study of a student who visualized the Marching Squares algorithm.

2 Background & Related Work

While different styles of visualization exist, our approach focuses on explanatory techniques. In this paper we focus on a third year Computer Graphics & Visualization course for Computer Science undergraduate-major students. These students are studying for their three-year BSc honors degree in the UK. In Semester 1, learners perform all the research and planning stages, and in Semester 2 they start coding, reflect on their code and demonstrate their solutions. The advantage of using *active learning* [5] techniques is that learners more proactively interact with the material engaging multi-sensory learning, rather than merely listening passively to the teacher. Students can become motivated by the task, if the task is something interesting and they see it being worthwhile. Grissom et al. [1] write "*The true value of using visualizations may lie not in their content but rather in their serving as a motivational factor to make students work harder*".

Active learning techniques are not new, e.g., Stasko [7] used animation as a learning aid, and Hundhasuen and Douglass [2] evaluate the benefits of using visualizations to help students learn algorithms, whereas tools such as Greenfoot (greenfoot.org) use animation to teach programming skills. But, most prior work only focuses on the (i) design and (ii) implementation of these algorithms. We, however, have developed a formal structure, that draws upon Kolb's [3] experiential learning cycle. We integrate *thinking*, with *modifying* (through improvements from formative feedback). *Integrating*, *experiencing* and *practicing* the ideas through sketching-by-design, building the tool, and *reflecting* on the work. Furthermore, our approach takes the student through the entirety of the Blooms taxonomy, from knowledge to synthesis and evaluation.

3 Methodology & Case study

The 20 credit computer graphics & visualization course is split over two teaching Semesters (before and after Christmas). In Semester 1, the students are first given traditional lectures on all the algorithms, choose their topic and started planning their visualization. Forty students chose from a list of 50 topics, including Marching Squares/Cubes, Volume Rendering, Ray-tracing, linear interpolation and use of transfer functions. Because each student investigates their own algorithm, they can readily work together and exchange design ideas without problems of plagiarism. Students, in Semester 2, develop their code and complete the assessment.

Our formal structure follows six parts (see Fig. 1 and explained below). At each stage, their work is evaluated formatively (with the teacher giving individual verbal feedback in a classroom setting), learners improve this version and submit for summative assessment. We include examples from a student (Tom) who chose the Marching Squares 2D piecewise linear contour algorithm.

Research. Students look at books, papers from IEEE and ACM digital libraries and other resources online to understand the algorithm or technique. Tom looked at the original Lorensen & Cline [4] algorithm and other papers.

Summary-document. A two-page summary of the algorithm is created. Every student has the same structure of: History, Pseudo

*e-mail: j.c.roberts@bangor.ac.uk
†e-mail: j.jackson@bangor.ac.uk
‡e-mail: cheadleand@lincoln.ac.uk
§e-mail: p.ritsos@chester.ac.uk

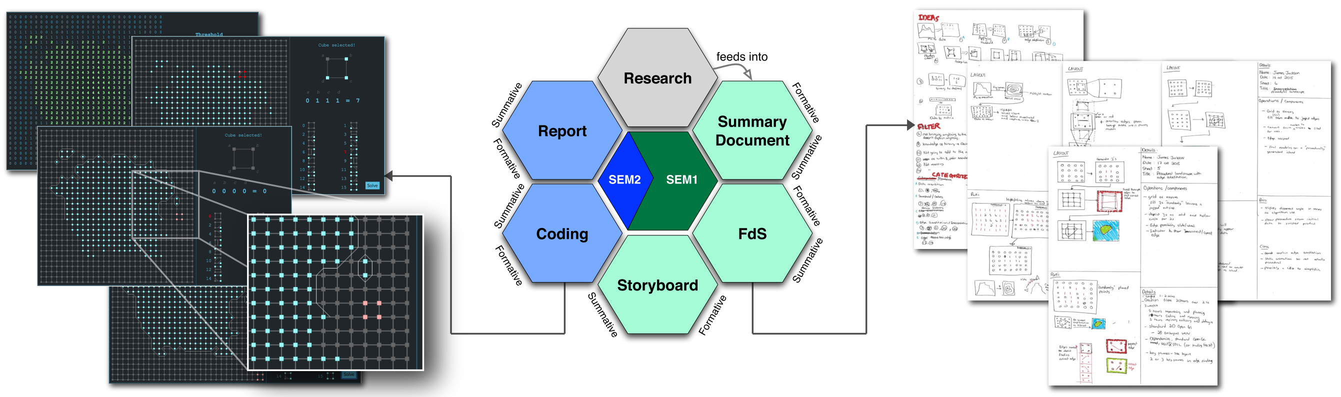

Figure 1: Our Explanatory Visualization active learning cycle. Students choose an algorithm, perform research, plan through sketching, gain formative and summative marks and demonstrate their tool that demonstrates a computer graphics or visualization algorithm.

code, Maths, Diagram, Application, Similar techniques, References. Several indicators of quality exist here, from written descriptions to the quality of the diagram. One diagram from Tom's two-page summary report is shown in Fig. 2 clearly articulating the main parts of the algorithm.

Design-by-sketching. Students use the Five Design-Sheet methodology [6] to sketch different visualization concepts (sheet 1), three alternative but potentially implementable designs (sheets 2,3,4) and sketch the realization design (sheet5). Formative feedback is given to the students on their ideas. Tom explored 23 ideas (on sheet 1), including: applying a threshold, matching the case in the look-up table, interpolating the piecewise segments to their exact position after retrieving them from the lookup-table, the act of marching from one voxel to another, resolving ambiguities, to generating statistical analysis over different cases from given data.

Storyboard. The learners create a storyboard to describe the main key stages of their explanatory visualization. This further confirms their knowledge of the main aspects of the algorithm.

Code. The students had to use OpenGL or WebGL due to prerequisite course requirements. However, other graphics libraries could be readily used instead. Over a 3 month development period, students met weekly with the academic tutor to gain feedback on their progression. Tom chose to develop an interactive, WebGL application. Users can generate test data to visualize (he spent effort to make it random, but to filter out very small unconnected scalar points), and then animate the index and contour generation per voxel.

Technical report. Finally, students wrote a technical report on their work, which included a critical evaluation of their achievements and explanatory visualization, and they gave a demonstration of their visualization to the tutor.

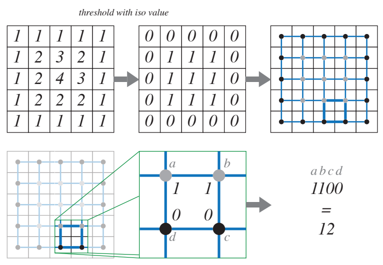

Figure 2: Marching Squares figure from Tom's two-page summary sheet, that explains the main stages.

4 Discussion & Conclusions

We have made a preliminary evaluation of the process. Students completed an anonymous questionnaire of 10 questions. We received positive and encouraging feedback. One student wrote "it broke down a project into its logical steps (design – implementation – evaluation)", another "it is a good way to guide students from understanding, thinking and then implementing". The two-page summary sheet gets the students thinking about the algorithm, and the tutor can ascertain if a student does actually understand the algorithm! Extra help can be given, if a student is confused. Sketching using the Five Design-Sheet [6] method encourages the students to be creative and imaginative. While students were given the opportunity to re-submit each FdS only two students decided to re-submit. On reflection, we believe that students would benefit from more design teaching in their computer science program, and using the FdS multiple times would benefit the students' learning. Previous years, we have used a more ad hoc method of design and building in this module. Our 6-part structure developed better solutions, and the grades were (on average) better than previous years. No plagiarism was found and students benefited from being able to discuss their work with others: "you have the opportunity to . . . explain the algorithm you are creating with students in the same course". We recommend other educators to use a similar *active learning* strategy and to use explanatory visualization in learning. because students perform better, and understand the work more deeply.

References

[1] S. Grissom, M. F. McNally, and T. Naps. Algorithm visualization in CS education: Comparing levels of student engagement. In *Proc. ACM Symp. Soft. Vis.*, pages 87–94. ACM, 2003.

[2] C. Hundhausen and S. Douglas. Using visualizations to learn algorithms: should students construct their own, or view an expert's? In *IEEE Symp. on Visual Languages*, pages 21–28, 2000.

[3] A. Y. Kolb and D. A. Kolb. Learning styles and learning spaces: Enhancing experiential learning in higher education. *Academy of management learning & education*, 4(2):193–212, 2005.

[4] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *SIGGRAPH*, pages 163–169, New York, NY, USA, 1987. ACM.

[5] C. Meyers and T. B. Jones. *Promoting Active Learning. Strategies for the College Classroom.* ERIC, 1993.

[6] J. C. Roberts, C. Headleand, and P. D. Ritsos. Sketching designs using the five design-sheet methodology. *IEEE Trans. on Vis. and Comp. Graph.*, 22(1):419–428, Jan 2016.

[7] J. T. Stasko. Using student-built algorithm animations as learning aids. *SIGCSE Bull.*, 29(1):25–29, Mar. 1997.