Research Journal

comp110-journal

1506919

November 27, 2016

1 When does a physical system compute?

This journal looks into how we define a computation device and also how we know when a computation is happening? Copeland distinguishes a computation as executing an algorithm [1]. Physical computation is defined as a physical system performing internal interactions, which consists of abstract entity, computation input and produces an output. We should feel confident that the output is correct. With this theory, the authors use a series of computing diagrams that can be used on many different systems, e.g. a computer abstract entity refers to its programming in a physical system. Other examples include a computing system which must also be capable of encoding, decoding and embedding. The article then goes into greater depth and discusses how inputs are broken down into smaller operations in standard gate-based computers, which is what this journal concentrates on. Furthermore, physical computational devices are capable of changing high-level assembly language into low-level assembly language (1, 0) without human input. Though computational output can change depending on the words (such as AND, OR, NOT) used to program a process. The main factor that defines a computational device is the ability to encode and decode, without this there

is no computation and it would just be considered a physical system. Computational devices are also used to simulate anything using virtual environments [2].

In conclusion, this paper highlights the fact that everyone has a different idea of what a computational device is and this journal concentrated on the framework of the devices, whilst also stating that a human could themselves be a computational entity. I agree about the importance of encoding and decoding in computing in regards to defining a computational device.

2 Experimental investigations of the utility of detailed flowcharts in programming

As the title of this article makes obvious it discusses flowcharts in relation to computer programming, since computers were first created flow charts have been used to describe and help explain computational processes, as programming languages became more complex as did the flowchart, becoming more structured [3], but even so more programmers argued if flowcharts were useful or hindering the way programming was taught. Previous experimental research found those who used flowcharts were better at knowing the parts needed to make a programme but not understanding other tasks. The authors did their own set of experiments five in total, in which students were beginning to learn programming were divided into a group using flowcharts and a group that wasn't. Experiment 2 studied how flowcharts helped the student compose a program to help solve a certain problem; the flowchart group did just as well as the non-flowchart group. Experiment 2 looked into if flowcharts helped with programming comprehension, the subjects were required to determine the values printed for various inputs and to trace the flow of execution, again there was no notable difference between the flowchart and non-flowchart group. Experiment 3, the students are no longer novices and the flowcharts were used to help with debugging and comprehension, the results of this

experiment showed it depended very much on the persons experience with flowcharts on whether it helped or not with their work but the difference between the flowchart and non-flowchart group was not a very significant one. Experiment 4 second year students used flowcharts to help them modify a programme, again no great difference in results of these that received a flowchart and those that did not. Experiment 5 investigated detailed flowcharts in a comprehension task, the subjects were amateur programmers, the non-flowchart group preformed the best but not by a significant margin. Looking at the results of all the experiments the non-flowchart group did better but because there was significant difference flowcharts might just be harder to understand instead of programs which the non-flowcharters were using. The paper concludes that flowcharts are just to outdated to help with detailed programming tasks but further research should be done with professionals or nonprogrammers [4].

I have some experience of this because when I was in my second year of secondry school in 2006 they were only just introducing programming to our school and we learnt how to programme a set of traffic lights using flowcharts, I enjoyed the whole process because the flowcharts made it simple and easy to understand, although this paper found flowcharts hindered programming with university level students, perhaps if they tried the same activities with 11 - 12 year olds the results may have been leaning more towards flowcharts than detailed programs.

3 A fast procedure for computing the distance between complex objects in three-dimensional space

This paper discusses collision detection in 3D and proposes a mathematical equation using shape models as a way of calculating distance between 2 objects, when the objects often changes position. For 2D collision problems Schwartz proposed using the O($log^2 M$) equation [5]. Unfortunately 3D has not been widely experimented on and Gilbert,

Johnson and Keerthis algorithm reducing computational time. The paper goes on to discuss how this can be applied to nonconvex objects. Section 2 of this paper shows the algorithm and applies it to a 3D object to help the reader understand how to calculate distance. Section 3 informs the reader of the preliminaries needed to be included in the algorithm to make it work. Section 4 shows the steps the algorithm takes to work out distance when being run. Section 6 changes the algorithm to account for the possible errors between the approximated value and the exact value. The algorithm was then tested on the types of example, a large gap between the objects, objects just touching, then objects intersecting, collecting the results and comparing them against similar algorithm tests [6].

My knowledge of maths is very limited, and all the programs I have used, already have collision built in so there is no need to do any hard algorithms. This paper went into great detail of the maths needed for collision and their ideal audience must have great understanding of degree level maths but as many coding programs do have collision built in I wonder if this paper has any relevance for todays programmers.

4 Go to statement considered harmful

In Letters to the Editor, go to statements considered harmful, the go to statement is blamed for decreasing the quality of programmers. The more go to statements there are in a program the more inadequate the programmer is considered to be. It is important that the program code works efficiently after it is written without extending computational time unnecessarily. The editor then proposes coding situations and language that can be used instead of the go to statement. The main reason the go to statement is considered harmful seems to be because it is hard to follow in the coding as the program is being run.[7]

This article is biased against go programs without providing much of an opposing argument. Donald E. Knuth writes a paper providing views both for and against the

abolishment of the go to statement [8], before providing an overall view. This makes Knuths paper more trustworthy than Letters to the Editor and helps the reader develop their own opinion.

5 Molecular computation of solutions to combinatorial problems

This paper discusses using molecules to compute, taking inspiration from Richard Feynmans idea of sub-microscopic computers. The paper then goes on to discuss algorithms to find if a graph has a Hamiltonian path or not, and the different molecules being tested. The molecules purpose was to encode the Hamilton path and the results were recoded [9].

Some in depth knowledge of chemistry would be required to follow this paper. The author did prove that molecules could be made to compute, but would require a lot more than one to rival the super computers of today, it would be interesting to see their application in robotics, after it was proven fungus can be used to control a robots movement [10], but doubt this knowledge would be used for robots in the home.

6 Experimental testing in programming languages, stylistic considerations and design techniques

[11]

References

[1] B. J. Copeland, "What is computation?" *Synthese*, vol. 108, no. 3, pp. 335–359, 1996.

[2] C. Horsman, S. Stepney, R. C. Wagner, and V. Kendon, "When does a physical

system compute?" in *Proc. R. Soc. A*, vol. 470, no. 2169. The Royal Society, 2014, p. 20140182.

[3] I. Nassi and B. Shneiderman, "Flowchart techniques for structured programming," *ACM Sigplan Notices*, vol. 8, no. 8, pp. 12–26, 1973.

[4] B. Shneiderman, R. Mayer, D. McKay, and P. Heller, "Experimental investigations of the utility of detailed flowcharts in programming," *Communications of the ACM*, vol. 20, no. 6, pp. 373–381, 1977.

[5] J. T. Schwartz, "Finding the minimum distance between two convex polygons," *Information Processing Letters*, vol. 13, no. 4-5, pp. 168–170, 1981.

[6] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi, "A fast procedure for computing the distance between complex objects in three-dimensional space," *IEEE Journal on Robotics and Automation*, vol. 4, no. 2, pp. 193–203, 1988.

[7] E. W. Dijkstra, "Go to statement considered harmful," in *Software pioneers*. Springer, 2002, pp. 351–355.

[8] D. E. Knuth, "Structured programming with go to statements," *ACM Computing Surveys (CSUR)*, vol. 6, no. 4, pp. 261–301, 1974.

[9] L. M. Adleman, "Molecular computation of solutions to combinatorial problems," *Nature*, vol. 369, p. 40, 1994.

[10] H. Schwarz, P. Rüger, A. Kicherer, and R. Töpfer, "Development of an autonomous driven robotic platform used for ht-phenotyping in viticulture," *Mech. Eng. Lett. Szent István Univ*, vol. 10, pp. 153–160, 2013.

[11] B. Shneiderman, "Experimental testing in programming languages, stylistic considerations and design techniques," in *Proceedings of the May 19-22, 1975, national computer conference and exposition*. ACM, 1975, pp. 653–656.