In P. Romero, J. Good, E. Acosta Chaparro & S. Bryant (Eds). Proc. PPIG 17 Pages 49 - 61

Effects of Experience on Gaze Behavior during Program Animation

Roman Bednarik, Niko Myller, Erkki Sutinen, and Markku Tukiainen

Department of Computer Science, University of Joensuu,
P.O. Box 111, FI-80101 Joensuu, FINLAND
firstname.lastname@cs.joensuu.fi

Abstract. The purpose of program visualization is to illustrate some aspects of the execution of a program. A number of program visualization tools have been developed to support teaching and learning of programming, but only few have been empirically evaluated. Moreover, the dynamics of gaze behavior during program visualization has not been investigated using eye movements and little is known about how program animation is attended by learners with various levels of experience. We report on an empirical study of the gaze behavior during a dynamic program animation. A novice and an intermediate group, a total of 16 participants, used Jeliot 3, a program visualization tool, to comprehend two short Java programs. Referring to previous literature, we hypothesized that the performance as well as the gaze behavior of these two groups would differ. We found statistically significant differences in performance measures and in fixation durations. Other commonly used eye-tracking measures, the fixation count and the number of attention switches per minute, seem to be insensitive to the level of experience. Based on the results, we propose further directions of the research into gaze behavior during program visualization.

1 Introduction

Program visualization is used to illustrate visually the run-time behavior of computer programs. These systems can be utilized, for example, in programming courses to support teaching of programming concepts to novice programmers. Jeliot 3 is an interactive program visualization system that automatically visualizes data and control flows of Java programs. It has been successfully used in classroom settings to teach programming to high school students [1].

Although several program visualization tools exist, only few have been evaluated and little knowledge is available about the aspects of gaze behavior during a dynamic program visualization. It is not clear how different users attend the animation and what cognitive efforts they have to exercise in order to comprehend the dynamic visualization. Therefore, in order to improve program visualization systems to fit their users best, it is a crucial issue to investigate the visual attention paths of users while visualizing a program. If a purpose of program visualization is to support the novices in their understanding, it is reasonable to study how their behaviors differ from the behaviors of intermediates. In other domains, eye-movement tracking has been successfully applied to investigate the gaze patterns of participants while performing their tasks. However,

no eye-movement based analysis of the gaze behavior during a dynamic program visualization has been conducted yet.

We report on an initial study in which we have employed a remote eye tracker to measure the gaze behavior of programmers during program comprehension facilitated by an animation tool, Jeliot 3.

The rest of the paper is arranged as follows. In Section 2, we review some related work in eye tracking research and program visualization, and Jeliot 3 is introduced. The experiment and results are described in Sections 3 and 4, respectively, and discussed in Section 5. Conclusions and future work are presented in Section 6.

2 Related Work

2.1 Eye Tracking

Humans move their eyes in order to bring an inspected object or a portion of it onto fovea, the high-resolution area of retina. This way the visual attention is closely linked with the direction of the eye-gaze, and most of the time it is also diverted to the point of visual inspection. Following this assumption, if we can track the movements of eyes, we can also get insights into and investigate the path and focus of attention during a task such as program comprehension. Furthermore, knowing which objects have been visually inspected and in which order and context, we can attempt to infer what cognitive processes were involved to perform the task related to these objects.

Eye tracker is a device that records eye movements. Most of the current eye trackers use infrared light emitters and video image analysis of the corneal reflections and pupil center to relate them to the direction of gaze. Typically, the accuracy of current eye trackers ranges around 1 degree, while the data is sampled at rates of 50–500Hz. Current eye trackers are relatively cheap and able to reliably and unobtrusively collect gaze data.

From the signal obtained from an eye tracker, two most important types of eye movements are usually identified: saccades and fixations [2]. *Saccades* are rapid ballistic movements of eyes that are executed to reposition the eyes from one location of attention to another one. A single saccade can last between 30 and 120 ms, can span over 1 to 40 degrees of visual angle [2], with velocities ranging up to 500 degrees per second [3]. No visual information is extracted during a saccade, a phenomena called saccadic suppression [4]. *Fixations* are eye movements stabilizing the image of an object on the retina. Typical fixation duration ranges between 200–300 ms [3]. It is assumed that during the period of a single fixation the information is extracted, decoded, and interpreted. The fixation duration can be therefore thought to be related with a required processing to extract and interpret the information [5, 6]. An accurate measurement and analysis of eye movements in terms of saccades and fixations provide researchers with the details of cognitive processing and related visual attention allocation within a performed task. For instance, the fixation count or sum of fixation durations on a certain element can be related to the importance of the element. In the context of program visualization interfaces, the relative fixation count measure can correspond with the relative importance of a representation (e.g. a code or a state diagram) of a program.

It is a well-known fact that eye movement patterns of experts and novices differ. Previous eye movement studies in other domains than program visualization have shown,

for instance, that (1) search strategies differ between novice and expert radiologists [7], (2) expert-pilots' eye movement patterns were better defined and the dwell times were significantly shorter than those of novices [8]. A common denominator in these and other reports is that domain knowledge and experience of participants seem to be the main factors influencing not only the performance, but also the related gaze behavior.

Visual attention tracking during program comprehension has been previously studied by Crosby and Stelovsky [9]. They used an eye tracker to discover the relationship between cognitive styles and individual differences, and code-reading patterns. In their study, novices and experts were eye tracked during an algorithm comprehension. However, only one representation of program was used (the code) and the focus of the research was mainly on the critical, but surface features of code, not on the behavior during a dynamic program visualization.

In the direction of investigating issues such as visual attention switching or a multiple-representation use during program comprehension or debugging, previous studies involved only a static precomputed stimuli and the analysis was based on a recording of mouse movements over a blurred interface [10, 11]. The validity of such an approach was shown to be questionable [12, 13]. To our knowledge, no eye movement based analysis of behavior during program animation has been conducted yet. This is certainly surprising, considering the importance of knowledge how the visual attention and cognitive processes involved in program comprehension are influenced by program animation.

2.2 Program Visualization

A number of program visualization systems have been developed over the previous years to teach programming or to visually debug programs. Here we will briefly review those systems that in some aspects are similar to Jeliot, the program visualization tool employed in the present experiment.

Javavis [14] is a tool that visualizes automatically the runtime behavior of the Java programs. It shows changes in the state of the program during execution using animated UML-like object and sequence diagrams. DDD [15], a debugging front-end, uses diagrams to illustrate the references between data structures during program execution. The diagram can be seen as graphs where nodes are the separate data structures (e.g. struct in C) and vertices are the references between them. The DDD does not explicitly visualize the control flow of the program. Jive [16] uses a similar approach to Javavis and DDD to visualize the program state using diagrams. The references, primitive values and variables are visualized similarly in Jeliot 3 and these systems. However, only Javavis visualizes control flow, but in less detail compared to Jeliot 3.

PlanAni [17] is a program visualization system that illustrates the data flow of a program during its execution. The use of variables in different purposes is illustrated through the roles of variables. The expression evaluation and control flow are also visualized. Currently, the animations must be programmed beforehand by an instructor and the visualization of object-oriented concepts is not supported. The organization of the user interface in PlanAni is similar to Jeliot. However, Jeliot does not visualize the roles of variables as PlanAni and PlanAni does not visualize the control flow.

2.3 Jeliot 3

Moreno at al. [18] have developed a program visualization system, called Jeliot 3. Its predecessor, Jeliot 2000, has been successfully used to improve the teaching of introductory programming helping the novices to acquire vocabulary to explain programming structures and concepts [1]. Jeliot 3 retains the novice-oriented GUI and animation display of Jeliot 2000. Jeliot 3 introduced a new design in order to make the system extensible and to allow for adding new features into the visualization. It visualizes automatically the execution of user-written Java programs by illustrating the data and control flow and object-oriented features of the program. Jeliot 3 can visualize a large subset of novice-level Java programs (see `http://cs.joensuu.fi/jeliot/`). The user interface of Jeliot 3 is shown in Figure 1.

The interface consists of four discrete areas. A code editor on the left hand side shows the program code, and during program visualization, the currently executed statement or expression is highlighted. A control panel in the bottom left corner is used to control the animation with VCR-like buttons. The largest area of the user interface of Jeliot is occupied by the visualization view showing the execution state of the program on the right hand side of the window. Visualization consists of method frames, local variables, expression evaluation, static variables, objects and arrays. Finally, an output console lies in the bottom right corner of the window, showing the output of the executed program. To sum it up, Jeliot provides four different areas of interest to the user: code view, animation view, control panel, and output console. Moreover, animation view is further divided into four different areas of interest: method, expression evaluation, constant, and object and array areas. Furthermore, there are separate specialized visualizations where only the call tree of the program or the execution history are shown.

In a typical session with Jeliot, a user either writes or loads a previously stored program. User can compile the program through the user interface of Jeliot. When the program is compiled, a visualization view, where the user can see the animation of the program execution, is opened. Jeliot shows the execution either step by step or continuously. User can control the speed of the animation and stop or rewind the animation at any point. User can select the current visualization with the tabs on top of the visualization view.

3 Experiment

The present research investigates the differences in the gaze behavior during program animation of participants with different levels of programming experience. Based on the results from available literature, our hypothesis was that the performance and gaze behavior of novices and intermediates differ during the program animation. In other words, our aim was to answer the question, whether intermediates and novices pay attention to the animation in a similar or different way. Our hypothesis is not surprising, since we naturally assume that a different level of experience shall result into a different gaze behavior and performance, as it has been found in other domains. More experienced programmers are expected to form better hypotheses about the problem and this knowledge should guide them to use the available representations in a distinct

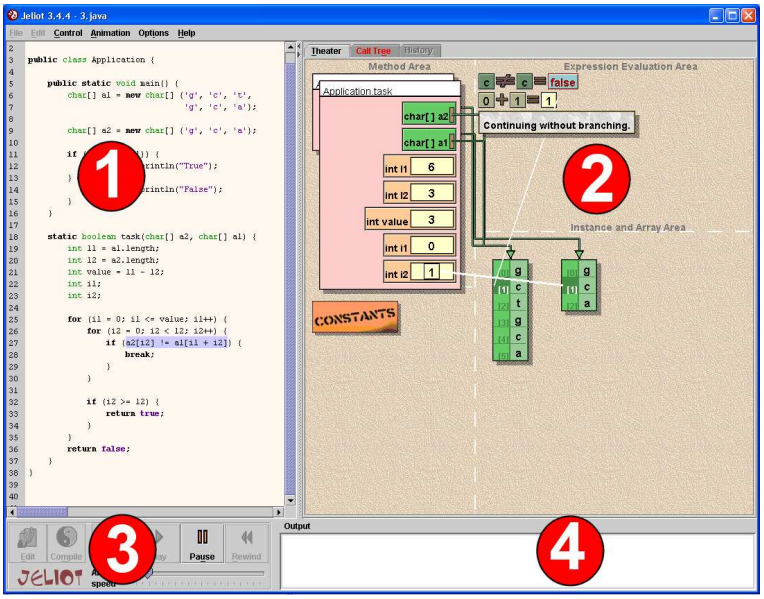

Fig. 1. User interface of Jeliot 3. Area 1 is code editor, area 2 is animation frame, area 3 is control panel and area 4 is output console.

way, compared to novices. We had a further assumption that novices would rely more on the visualization than code and the other way around for intermediates.

To validate these hypotheses, we conducted an empirical experiment where we used a remote eye-tracker to record the gaze behavior of the participants during program comprehension task aided by an animation. Two groups of participants with different level of experience used Jeliot 3 to comprehend three short Java programs while their eye movements were simultaneously tracked.

3.1 Method

We used a between-subject design with experience (novice or intermediates) as the factor. The depended variables were: relative fixation count over the areas of interest, number of switches per minute and mean fixation duration over the areas of interest and in overall. The fixation count is a measure related to the level of participant's interest in an area. The number of switches per minute is a measure of attention allocation dynamics. The mean fixation duration is associated with the depth of processing required to understand an attended element. Only the gaze data during the program animation were used in this analysis because that is the only time when all the representations were available concurrently and the selection of the attended representation would make a difference in understanding the program. Most of the analysis was carried out using ANOVA and planned comparisons based on t-test.

3.2 Participants

Eighteen participants were recruited from high-school students attending a university-level programming course, undergraduate and graduate computer science students from local university. Due to technical problems with the eye tracking, data from two participants had to be discarded. Therefore, the results are based on the data collected from 16 subjects (13 male, 3 female). Participants were divided into two groups according to their level of programming experience. Participants with less than 24 months of programming experience were regarded as novices and above 24 months as intermediates. The characteristics of the two groups are presented in Table 1. Groups' mean values for programming experience (in months) and Java experience (in months) and counts for previous experience with Jeliot 3 (yes=1, no=0) and previous experience as professional programmer (yes=1, no=0) are shown. Standard deviations are shown in parentheses.

Table 1. Characteristics of the groups. * marks a significant difference between groups in two-tailed t-test (interval values) or χ^2-test (nominal values) with $p < 0.05$

Experience level	Count	Prog. exp.*	Java exp.*	Jeliot exp.	Prof. exp.
Novices	8	12.8 months (6.9)	6.4 months (4.6)	3	1
Intermediates	8	85.5 months (56.4)	19.8 months (15.0)	2	1

3.3 Materials and Apparatus

Three short Java programs, factorial computation, recursive binary search, and naïve string matching were presented to the participants. The lengths of the programs in lines of code were 15, 34, and 38 respectively. Each of the programs generated only one line of output and did not require any user input. The names of methods and variables were altered so that the recognition of a program based on these surface features would be difficult.

In our study, we used an adapted version of Jeliot 3 which logged all the user actions and all the changes in the visualization of the programs to be compared with the eye tracking data. However, this material is not used in this analysis. The specialized visualizations, the execution history and the call tree, were disabled to avoid problems in interpreting the gaze behavior.

The remote Tobii ET-1750 (sampling rate 50Hz) eye tracker making no contact with participants was used to track eye movements; the eye tracker is built into a TFT panel so no moving part is visible and no sound can be heard during the recording. Only a computer mouse was available during the experiment to interact with the tool. The interaction protocols (such as mouse clicks) were collected for all the target programs, and audio and video were recorded for a whole session. Fixations shorter than 100 ms were disregarded from analysis. We have defined four main areas of interest matching

the four main areas in the Jeliot interface: the code, the animation, the control, and the output area. Figure 2 illustrates the experimental settings used in the study.

Fig. 2. Experimental settings.

3.4 Procedure and Design

The experiment was conducted in a quiet usability lab. Participants were seated in an ordinary office chair, near the experimenter, and facing a 17" TFT display. Every participant then passed an automatic eye-tracking calibration. During the calibration procedure, a participant had to follow sixteen shrinking points appearing one by one across the screen. If needed, the calibration was repeated in order to achieve the highest possible accuracy.

After a successful calibration, participants performed three sessions, each consisting of a comprehension phase using Jeliot 3 and a program summary writing phase. Participants were instructed to comprehend the program as well as possible and they could interact with Jeliot as they found it necessary. The target programs contained no errors and were always preloaded into Jeliot and compiled. The duration of a session was not limited.

The first program was factorial computation and it was used as a warm-up and the resulting data were discarded. The order of the two actual comprehension tasks was randomized so that half of the participants started with the recursive binary search and other half with naïve string matching.

4 Results

4.1 Completion and Animation Times

Mean completion times for the comprehension phase were 17.6 minutes (SD= 10.0) for novices, and 9.8 minutes (SD=2.6) for intermediates; the difference was statistically significant according to a two-tailed t-test $(t(7) = 2.48, p < .05)$. From that time, novices spent on average 85.4% (SD=9.6) animating the program whereas intermediates spent 52.9% (SD=20.0) of their time to animation; the difference was statistically significant according to the two-tailed t-test $(t(7) = 5.38, p < .01)$.

4.2 Fixation count distribution

Figure 3 shows a relative fixation count distribution over the areas of interest during the animation. Both groups spent most of the viewing time fixating the animation area, 57.4% (SD=11.9) novices, and 54.8% (SD=15.2) intermediates, of all fixations during the program animation. Next, 39.4% (SD=11.2) and 43.3% (SD=14.5), novices and intermediates, respectively, of all fixations was paid to the code area. No significant effect of experience on the distribution of fixations was found, without any interaction between the area of interest and experience. The fixation count has significantly differed between all four areas of interest, $F(3, 42) = 105.75, p < .001$. The planned comparison revealed a significant difference in the fixation count between the two most attended areas, the code and the animation $(t(15) = 2.29, p < .05)$.

4.3 Switching Behavior

Figure 4 illustrates the switching behavior as expressed by the number of switches per minute between the different areas of interest. The average number of switches per minute was 30.15 (SD=10.66) and 27.57 (SD=8.04) for novices and intermediates, respectively. The analysis of the effect of experience on the switching behavior discovered no significant change in the number of switches per minute, $F(1, 14) = 0.004, ns$. The switch between the code and the animation areas was far most common, $F(5, 70) = 145.25, p < .001$. Finally, the interaction effect between type of switch and experience was not significant, $F(5, 70) = 0.421, ns$.

4.4 Fixation Durations

Figure 5 shows the mean fixation durations during animation for the four main areas of interest and the overall mean fixation duration. These have been computed as a sum of durations of all fixations landing at an area of interest divided by number of the fixations. Since the programs did not generate an extensive output, some of the participants were not gazing to this area of interest. For the analysis, the missing values were replaced by the mean value of a group.

The overall mean fixation duration was 406.49 ms (SD=81.40) and 297.26 ms (SD=80.52) for novice and intermediate group, respectively. The effect of area of interest on the mean fixation duration was nearly significant, $F(3, 42) = 2.79, p = .052$.

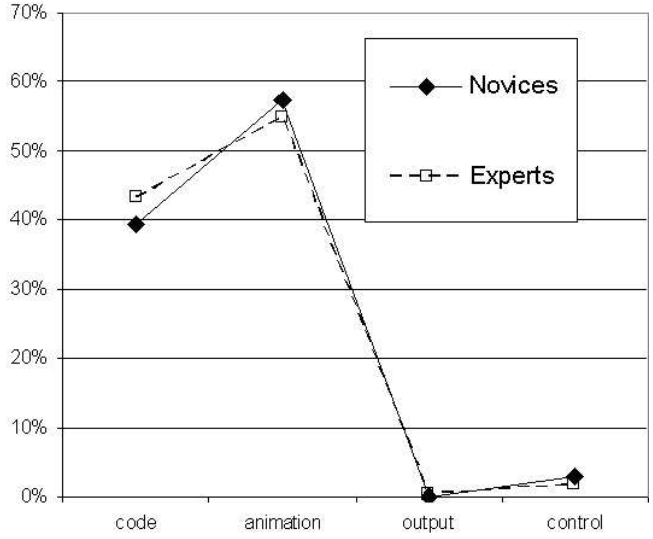

Fig. 3. Relative fixation count distribution during animation.

We also found an interaction between the fixation durations on the areas of interest and the level of experience, $F(3, 42) = 2.87, p = .048$. The effect of experience on the mean fixation durations was significant, $F(1, 14) = 8.98, p = .01$. Moreover, the effect of experience on overall fixation duration, $F(1, 14) = 7.16, p = .018$, was also significant.

5 Discussion

Intermediates completed the comprehension phase much faster than novices. Intermediates also spent significantly less time animating the programs which was in agreement with the hypothesis that intermediates would concentrate more on the code reading. This happened, however, only before they began and after they stopped visualizing the program. Both times can be kept as measures of performance. The initial code-reading episodes could have affected the behavior of the intermediates during the program animation compared to novices. Sajaniemi and Kuittinen [17] reported that during exercise sessions, students using PlanAni did not pay attention to the program code as much as to the visualization. Our results agree with this observation. Although both areas were attended with high fixation counts, it was more common to use the visualization than the code area during program animation, in our study, regardless the experience.

Analysis of the comprehension summaries have been done elsewhere in Bednarik et al. [19] with the program summary analysis by Good and Brna [20]. In this analysis, the summaries of intermediate subjects were found to be slightly better in the quality, but

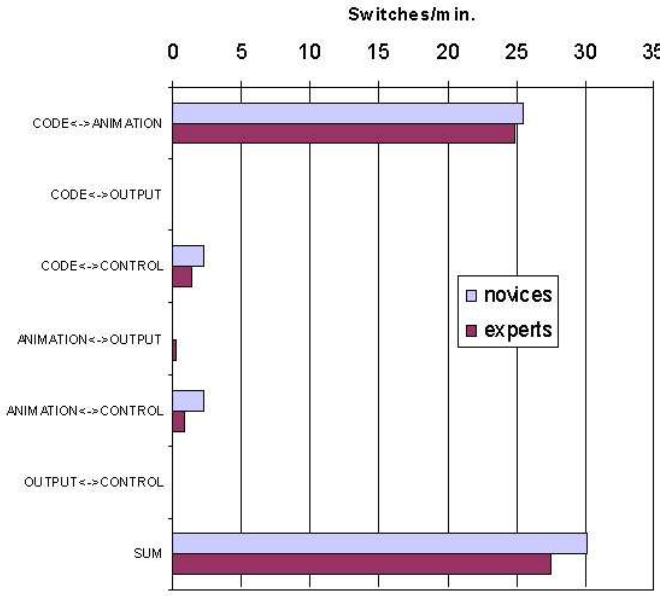

Fig. 4. Number of switches per minute between the main areas of interest.

there were no statistically significant differences found. Intermediates used higher level of abstraction than novices but again there were no statistically significant differences.

The results of this experiment related to the gaze behavior during program animation show that the relative fixation counts and the switching behavior between the areas defined in this study are insensitive to the level of experience. The distribution of fixations between code and animation was slightly more balanced for more experienced participants, but did not significantly differ from the distribution of novice fixations. With respect to these measures, we have to reject our hypothesis. Most of the animation time was spent on viewing the visualization part of the Jeliot interface.

The switches between code and animation areas were the far most common during the animation and therefore the sum of all switches is mostly composed by this type of switch. The code-control and animation-control switches were higher for novices. This is probably due to the fact that novices were interacting more with the tool during the animation than intermediates and therefore attending the control panel more often [19]. In terms of the total number of switches per minute, the two groups exhibited about the same behavior.

With respect to previous eye movement studies investigating the relationship between gaze behavior and expertise, these result are rather surprising. Several factors could, however, explain the results. One explanation seems to be that the features of animation attract equally novice and intermediate programmers to attend the animation in similar patterns. The visualization environment restricts the access to the elements of the graphical representation to only a short period of time, therefore the effects of

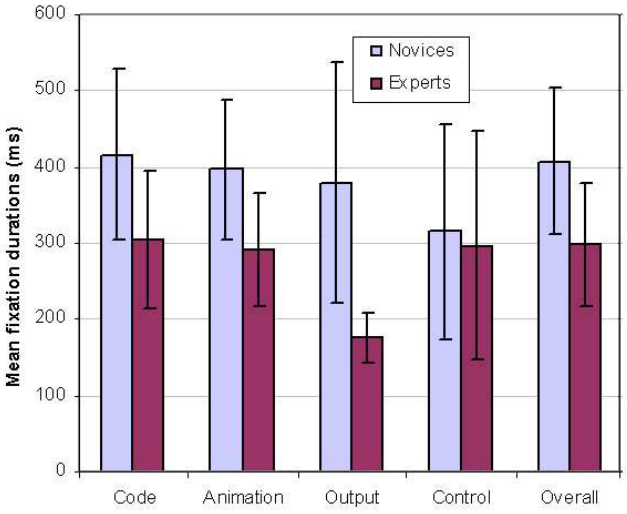

Fig. 5. Mean fixation duration during animation.

experience cannot materialize in the gaze measures used in this experiment. We also believe that more accurate measures have to be developed to reveal the differences between these two groups. For example, we could measure the disassociation between the current animation step and the gaze of the subject. Another possibility for not observing differences in the gaze behavior could be the number of subjects involved in the study and this will be taken into the consideration in the further studies. Finally, the gap between the skills of the two groups involved in this experiment might not be big enough to yield statistically significant differences in gaze behavior during animation.

Despite not finding differences in fixation count distribution and switching behavior, we did find a significant effect of experience on the mean fixation duration. For all the main areas (except for the control area) of the display and in overall, the mean fixation duration of intermediates was shorter than that of novices. This supports the results from previous studies and could be explained by at least two facts or a combination of both. One possibility is that, during the animation, intermediates might have an advantage of already formed hypothesis about the visualized problem. This hypothesis would be formed during the initial code reading before animating the program. The second explanation could be the available domain-knowledge and programming experience of the intermediates which would enable them to interpret the animation faster. From the mean fixation duration over the control panel, we can observe that novices and intermediates alike needed about the same time (300 ms) to decide what buttons they are going to use in order to control the flow on the ongoing animation.

Altogether, these findings could indicate that a difference in the programming experience can be seen in the mental efforts paid while attending the animation, while it does not affect the general patterns how the animation is attended. Both groups attend

the suggested attention loci in about same way, but the more experienced programmers extract the information faster and, most probably, are therefore able to pay attention to the surrounding context. When a consecutive attention switch is suggested by the animation, both groups will follow it and thus exhibit similar switching behavior.

6 Conclusion and Further Work

We have conducted an empirical experiment to discover the aspects of gaze behavior during the dynamic program visualization. We employed a non-intrusive remote eye tracking equipment to record the eye movements of programmers with various level of experience. Our results, in terms of the attention switches between different program representations and the distribution of fixations, show no difference in the gaze behavior between novice and intermediate group of programmers during program animation. In other words, the focus of visual attention seems to be distributed in time and space evenly regardless of the experience in programming. When the level of processing required to attend the animation is measured as a mean duration of fixations over the main areas of interest and in overall, our results show that novice programmers spend significantly more time on extracting the features of animated concepts. We propose this difference to be linked to the experience level and with a pre-established model of the algorithm being animated. The performance measures seem to support this hypothesis.

Our initial experiment provides a take-off mark for further studies investigating gaze behavior related to the dynamic program visualizations. Several directions for future research can be taken. Based on the general, macro-level patterns presented in this paper, we aim to deconstruct the behavior into more micro-level sequences. Between our next aims belong to investigate the effects of the discrete animation elements on the gaze behavior as well as the changes in the behavior in a course of time. Among the questions raised by the present study belong, what kind of suggested switches are consumed during the animation and whether the decision differs given the level of experience.

To answer the questions, we plan to develop a methodological framework for a reliable application of eye-movement tracking in the context of program visualization. These studies shall provide us with a deeper understanding about the cognitive processes involved in program comprehension during program visualization.

Acknowledgments

We would like to thank all participants for taking part in this study. We acknowledge Andrés Moreno for a help with preparation of this study.

References

1. Ben-Bassat Levy, R., Ben-Ari, M., Uronen, P.A.: The Jeliot 2000 program animation system. Computers & Education **40** (2003) 15–21
2. Sibert, L.E., Jacob, R.J.K.: Evaluation of eye gaze interaction. In: CHI 2000, ACM Press (2000) 281–288

3. Rayner, K.: Eye movements in reading and information processing: 20 years of research. Psychological Bulletin **124** (1998) 372–422
4. Matin, E.: Saccadic suppression: a review and an analysis. Psychological Bulletin **81** (1974) 889–917
5. Carpenter, P.A., Just, M.A.: Eye fixations during mental rotation. In Senders, J.W., Fisher, D.E., Monty, R.A., eds.: Eye movements and the higher psychological functions. Erlbaum, Hillsdale, NJ (1997) 115–133
6. Goldberg, J.H., Kotval, X.P.: Eye Movement-Based Evaluation of the Computer Interface. In Kumar, S.K., ed.: Advances in Occupational Ergonomics and Safety. IOS Press, Amsterdam (1998) 529–532
7. Nodine, C., Mello-Thoms, C.: The nature of expertise in radiology. In Beutel, J., Kundel, H., Metter, R.V., eds.: Handbook of Medical Imaging. SPIE Press (2000)
8. Kasarskis, P., Stehwien, J., Hickox, J., Aretz, A., Wickens, C.: Comparison of expert and novice scan behaviors during VFR flight. In: The 11th International Symposium on Aviation Psychology. (2001)
9. Crosby, M., Stelovsky, J.: Subject Differences in the Reading of Computer Algorithms. In Salvendy, G., Smith, M.J., eds.: Designing and Using Human-Computer Interfaces and Knowledge-Based Systems. Elsevier (1989) 137–144
10. Romero, P., du Boulay, B., Cox, R., Lutz, R.: Java debugging strategies in multi-representational environments. In: The 15th Annual Workshop of the Psychology of Programming Interest Group (PPIG'03). (2003) 421–434
11. Romero, P., Lutz, R., Cox, R., du Boulay, B.: Co-ordination of multiple external representations during Java program debugging. In: Empirical Studies of Programmers symposium of the IEEE Human Centric Computing Languages and Environments Symposia, Arlington, VA (2002) 207–214
12. Bednarik, R., Tukiainen, M.: Visual attention tracking during program debugging. In: NordiCHI'04, ACM Press (2004) 331–334
13. Bednarik, R., Tukiainen, M.: Effects of display blurring on the behavior of novices and experts during program debugging. In: CHI '05: CHI '05 extended abstracts on Human factors in computing systems, ACM Press (2005) 1204–1207
14. Oechsle, R., Schmitt, T.: JAVAVIS: Automatic Program Visualization with Object and Sequence Diagrams Using the Java Debug Interface (JDI). In Diehl, S., ed.: Software Visualization. Volume 2269 of Lecture Notes in Computer Science., Springer-Verlag (2002) 176–190
15. Zeller, A., Lütkehaus, D.: DDD — A Free Graphical Front-End for UNIX Debuggers. ACM SIGPLAN Notices **31** (1996) 22–27
16. Gestwicki, P., Jayaraman, B.: Interactive visualization of Java programs. In: IEEE Symposia on Human Centric Computing Languages and Environments. (2002) 226–235
17. Sajaniemi, J., Kuittinen, M.: Program animation based on the roles of variables. In: ACM symposium on Software visualization, ACM Press (2003) 7–16
18. Moreno, A., Myller, N., Sutinen, E., Ben-Ari, M.: Visualizing Programs with Jeliot 3. In: Advanced Visual Interfaces (AVI 2004). (2004) 373–376
19. Bednarik, R., Myller, N., Sutinen, E., Tukiainen, M.: Analyzing Individual Differences in Program Comprehension with Rich-Data Capture. Submitted (2005)
20. Good, J., Brna, P.: Program comprehension and authentic measurement: a scheme for analysing descriptions of programs. International Journal of Human-Computer Studies **61** (2004) 169–185