Journal for Geometry and Graphics
Volume 18 (2014), No. 2, 239–247.

How to Promote Student Creativity and Learning Using Tutorials in Teaching Graphics and Visualisation

Anders Hast

Dept. of Information Technology, Division of Visual Information and Interaction
Uppsala University, Lägerhyddsv. 2, 75105 Uppsala, Sweden
email: anders.hast@it.uu.se

Abstract. Course assignments play an important role in the learning process. However, they can be constructed in such a way that they prohibit creativity, rather than promoting it. Therefore it was investigated how programming assignments are set up that students encounter in computer science education and which approach could help the students in problem solving and whether these would help or prohibit them to be creative or not. Especially, an online tutorial about visualisation using VTK and Python was used as an example in different courses on visualisation. It was also examined how students in the computer graphics courses that did not have access to such tutorial answered questions about assignments.

Key Words: Tutorials, creativity, motivation, assignments, scientific visualisation and computer graphics programming

MSC 2010: 97U50, 97Q20

1. Introduction

Course assignments play an important role in the learning process and often allow the students to comprehend the theoretical part of the course by applying it in practice. However, programming assignments in general, also those in graphics and visualisation, can be constructed in such a way that they prohibit creativity. Instead, the task is merely limited to find out what the teacher expects, rather than actually promoting the learning process or increasing the students enthusiasm and the self confidence that comes from overcoming problems that are actually understood. Therefore it was investigated how programming assignments are set up that students encounter and which approach would be most beneficial for them. The idea was to examine, in particular, to what degree online tutorials could help the students in problem solving and whether these would help or prohibit them to be creative or not.

An online tutorial about visualisation using VTK and Python was used as an example in different courses on visualisation. The students had access to it during the assignments,

ISSN 1433-8157/$ 2.50

and it explains how to make scientific visualisations and what basic graphics algorithms are needed. The task was to solve problems that were described with pointers to the tutorial so that they could find the essential program examples, which would help them to solve the task. It was also examined how students in the computer graphics courses that did not have access to such tutorial answered questions about assignments.

2. Evaluating assignments in computer science education

Assignments in computer science education (CSE) can be constructed in many ways. However, three different approaches for constructing assignments were identified at forehand:

A. A *code skeleton* is provided where the teacher has removed some lines from a working program. The students task is to fill in the missing lines according to some specification.

B. A *working program* is provided. The students task is to add new functionality to that program according to a specification.

C. A *tutorial* with more or less complete programs describing different functionality is provided. The task is to write a program according to a specification. The students are free to use any program or functionality from the tutorial.

The courses involved in the evaluation presented in this paper used to have predominantly assignments of type **A**. The idea was that creativity would be notably promoted by changing them to type **B** or **C**. The hypothesis was that too many courses still have type **A** and that students would prefer to have assignments of type **B** or **C**. Espcecially, it was interesting to find out how type **C** assignments were perceived by the students. Subsequently, the computer graphics course was changed to have assignments of type **B** and the scientific visualisation course was changed to have type **C**. All assignments are found online at *studentportalen* [18]. Most of them contain a lot of explaining text, images and suggestions, so they are in fact a mini tutorial by themselves.

The *computer graphics course* [16], which gives 10 credits, uses the textbook by Angel and Shreiner [1], and 23 students in total answered the questionnaire. The students have to do three assignments and one larger project.

The *scientific visualisation course* [17] gives 5 credits and uses the textbook by Schroeder et al. [13]. Even if the assignments have been changed to be of type **C**, some have pointers to working code examples. Nevertheless it is expected that the students use the tutorial [19] to figure out how to do the required changes defined in the assignments. A total of 24 undergraduate students answered the questionnaire. An example page from the tutorial is shown in Figure 1, where it is explained how conversion of files to VTK format can be done.

The *graduate course on scientific visualisation* has been given in two variants, and a total of 16 students answered the questionnaire. The first was given as a two day workshop by UPPMAX [20] and the second was a graduate course given by SeSE [14], which have a syllabus very much like the under graduate course. However, the project is aimed at visualising the data that the PhD students use in their own research. UPPMAX is the 'Uppsala Multidisciplinary Center for Advanced Computational Science', which is a high performance computing (HPC) resource at Uppsala University. UPPMAX is one of six high performance computer centres in Sweden under SNIC [15], which is the 'Swedish National Infrastructure for Computing'. SeSE is a Swedish research school aiming at fostering collaboration between Swedish PhD students in different fields and giving courses in the field of HPC, visualisation,

Converting Data Files

Linked from VTK Tutorial, Reading Files

If you would like to convert the data you have into the *vtk* file format so that you can use your favorite vtk program to visualize it, then you can use a writer in vtk that will do the job for you. This short tutorial will show you how easy it is to convert data, in this case the data files used in How to Read Text Files. Please download the code Convert.py Don't forget to give the file names as an argument when you run the code:

```
python Convert.py data1.txt data2.txt data3.txt new_data.vtk
```

Contents [hide]

Explanation of the Code

Load all modules.

```
import sys
from vtk import *
```

Import the function **readPoints** from the ReadPoints.py file explained in: Reader for ASCII Files.

```
from ReadPoints import *
```

Create the unstructured grid and use your reader from the previous exercise.

```
data=vtk.vtkUnstructuredGrid()

data.SetPoints(readPoints(sys.argv[1]))
data.GetPointData().SetVectors(readVectors(sys.argv[2]))
data.GetPointData().SetScalars(readScalars(sys.argv[3]))
```

Use a Unstructured Grid Writer to write the data to the .vtk file. Yes, that is all you need to do!

```
Data=vtkUnstructuredGridWriter()
Data.SetInput(data)
Data.SetFileName(sys.argv[4])
Data.Write()
```

Exercises

Do the following exercises!

Visualization

Change the program BallsArrows.py so it will use your vtk file instead of reading the individual textfiles.

Convert from OBJ to VTK

Make a program that converts a .obj file to .vtk. Change the program Teapot.py so it will read the vtk file instead. You can actually use the vtkDataWriter, which is the parent class of vtkUnstructuredGridWriter

Figure 1: Screenshot from one web page in the VTK Tutorial that describes how to convert data ASCII files produced by Matlab® or any other program to standard VTK format. The code is explained in detail as well as how to run it. There are links to the source code and other pages explaining further details. Moreover, there are usually exercises and sometimes also answers.

big data and other related areas.

A questionnaire was filled in by the students, and the results from the students answers will be discussed. Moreover, conclusions will be drawn about how to proceed in the ongoing work to improve assignments in order to promote student creativity and learning.

3. Creativity, motivation and building new knowledge

Assignments can be constructed in such a way that the students feel that they have to figure out what the teacher intended and follow his line of thoughts, rather than being creative, exploring and getting new insight. This kind of assignments is typically not motivating for the students to learn more than just doing the task given them. Type **A** is a typical example of such assignments. The other types allow students and motivate them to use their gained skills and new knowledge to explore and discover further. By making use of the tutorial they are, in an indirect way, encouraged to also add more functionality than requested in the assignment. This is often easy to do in graphics and visualisation as the result is visual. By changing colours as well as appearance of objects, glyphs, or whatever is on the screen, the students can play around with the application and change it to their liking.

Creativity can be promoted in many ways. In both courses it was provided extra credit opportunities, as discussed by Roberts [11]. Furthermore, Romeike [12] identify three drivers for creativity in computer science education, namely: the person with his motivation and interest, the IT environment and the subject of software design itself. Obviously, they are all interconnected, and the aim was to improve the third driver, while in the same time taking advantage of the first one, by providing challenging and interesting tasks to perform, i.e., design an application that did something interesting.

It was noted by Knobelsdorf and Romeike [8] that *"The activities mostly start with gaming, followed by exploring applications and their possibilities, experimenting with the computer, knowledge gathering, and the Internet usage".*

This observation was based on work by Knobelsdorf and Schulte [9], where it is noted that students starts as users and then become designers, which is a natural development from simple to more complex activities. On-line tutorials typically allow students to do this as they can explore the tutorial and add functionality according to their interest and not only because the teacher say so.

The advantage with such e-learning materials is that students can build their own conceptual understanding and improve their skills. Faessler et al. [6] discuss the advantages of PBL based e-learning material. Nevertheless, Boyer et al. [3] point out that many have investigated the cognitive aspects [4] of e-learning, rather than motivational aspects. Obviously, motivation is important as the first driver identified by Romeike.

Furthermore, tutorials are important as they present models of how things can be done, i.e., solutions to simple problems or general cases that can applied by the students on their particular problem. VandegGrift [21] discuss different learning theories and what helps students to learn. She points out that students build knowledge by looking at such prototype models [10]. For courses that include programming this would typically be pseudo code or working code examples provided by the teacher. Similarly, the constructivist learning theory [2, 5] claims that students build knowledge based on existing knowledge and upon the foundation of previous learning [7]. VanDeGrift say that *"students must actively construct new ideas and experiment with those new ideas and mold them into existing knowledge".*

4. Results of the questionnaires

A questionnaire was handed out during one course on computer graphics for undergraduate students, one course in scientific visualisation, also for undergraduates and also in yet another scientific visualisation course, but this time for graduate students. The questions asked were:

1. *Which approach has been most common in programming courses you have taken?*
2. *Which approach do you prefer?*
3. *Which approach do you think will be most helpful for you to learn new things?*
4. *Which approach will allow you to be creative and try to invent new things?*

The result of Question 1 is illustrated in Figure 2. It is obvious that many students still do encounter questions of type **A**, just as was expected.

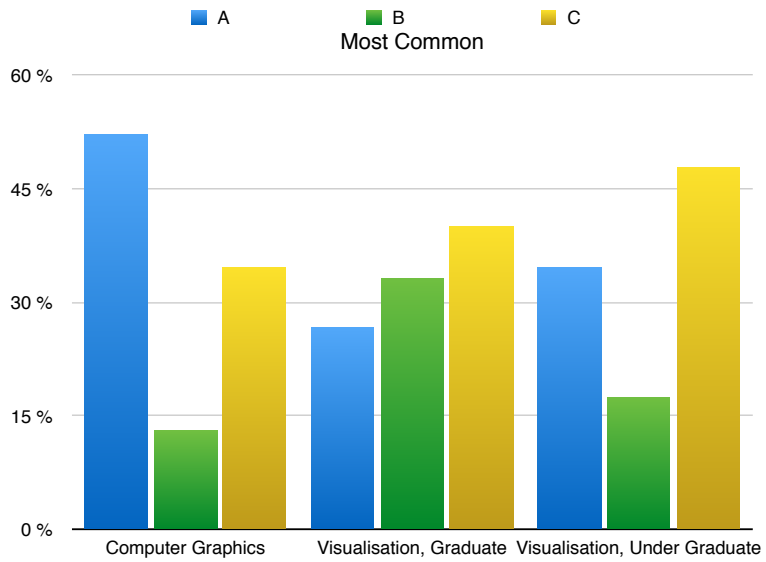

Figure 2: Outcome of Question 1: *"Which approach has been most common in programming courses you have taken?"*

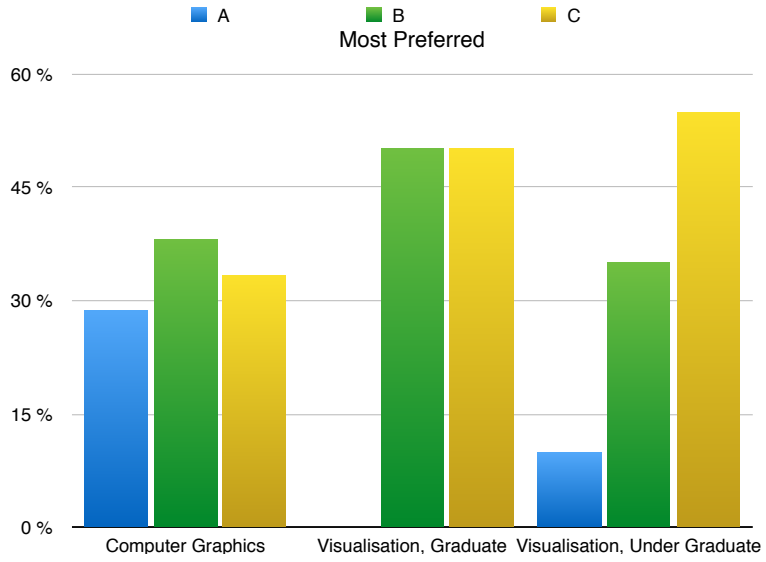

Figure 3: Outcome of Question 2: *"Which approach do you prefer?"*

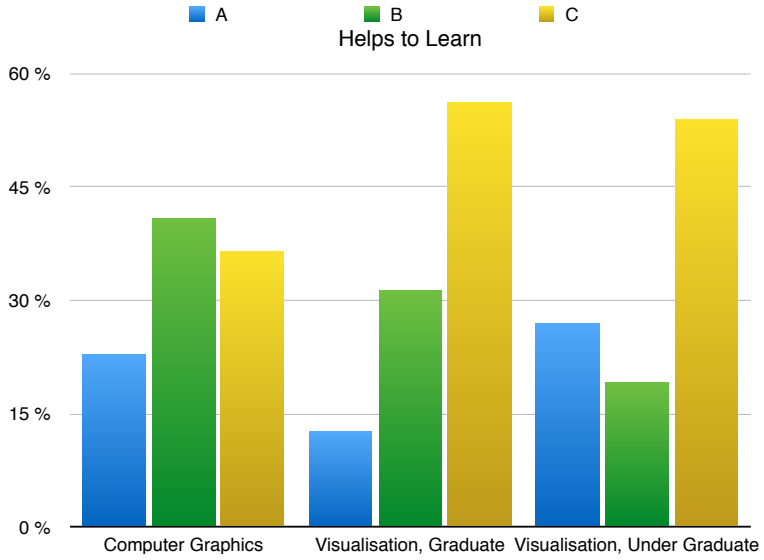

Figure 4: Outcome of Question 3: *"Which approach do you think will be most helpful for you to learn new things?"*

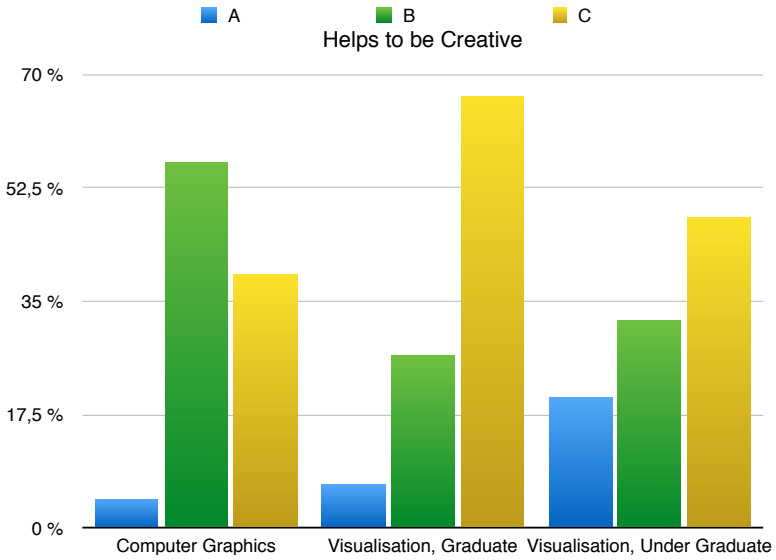

Figure 5: Outcome of Question 4: *"Which approach will allow you to be creative and try to invent new things?"*

The result of Question 2 is shown in Figure 3, and it can be noted that many computer graphics students actually prefer type **A**, but for the scientific visualisation course the types **B** and **C** are preferred by most students, while just a few advocate type **A**.

When asking which approach they think will be most helpful for learning new things (Question 3), there are some notable differences, as shown in Figure 4. Even if type **A** was preferred by the computer graphics students, they did not generally think it would help them learn new things. Likewise, type **C** was the most preferred in the scientific visualisation courses.

Even more computer graphics students realised that type **A** do not promote creativity. And once again the scientific visualisation students preferred type **C**, as can be seen in

Figure 5, where the result of Question 4 is shown in the graph.

Interestingly, more computer graphics students prefer type **B** over **C**. For the scientific visualisation students, on the other hand, the opposite is true. Even though the number of students asked does not allow us to draw general conclusions, this difference might depend on the fact that the computer graphics students encountered type **B** during the course, while the scientific visualisation students encountered type **C**.

5. Discussion

In retrospect, it was understood that it was unclear for some students what the differences between the types really were. The idea was that type **A** was not a working program since code was removed from it. Nevertheless, some students seemed to have thought it was the same as type **B**. Hence, some choose **A**, when they were supposed to choose **B** and vice versa. There is also a risk that students in the visualisation courses confused type **C** with **B** as some assignments gave pointers to a working program, even if the tutorial always was pointed out as the main source of information.

Moreover, the number of students answering the questionnaire was rather low (23, 16, 24, for each group, respectively). Hence, one should not draw decisive conclusions by looking at the bar graphs. Nevertheless, the trend is obvious, also from the authors own experience, that type **A** assignments are still quite common. Furthermore, it is also obvious that type **B** and **C** are generally preferred compared to type **A**.

6. Conclusion

The way assignments are designed can either prohibit or promote creativity. Still, many assignments contain non working code, where some lines are missing and the students are supposed to fill in that code. This type of assignments hardly allows students to be creative, instead they try to figure out what the teacher had removed. Moreover, they are not explicitly motivating the students to do more than just filling it the missing code.

Assignments based on working code, on the other hand, allows the students to explore the behaviour before adding new functionality. Playing around with the code, gives an initial understanding of what the code does, and this is an important first step in the creative process. Then it is up to the student how to solve the task given in the assignment. By having access to an online tutorial, the students can learn about how to solve such task in a generic way. They must themselves figure out how those solutions apply to their code. This might not promote creativity in itself. However, by reading through the examples and having access to exercises and solutions, the students will see what the possibilities are and hopefully they will be stimulated to try out things that they think can improve their code. This is especially true for assignments that produce visual results, which is always true for computer graphics and scientific visualisation.

It is interesting to note from the evaluation that the computer graphics students, which had not access to a tutorial in their course, generally were more positive to assignments that do not include a tutorial. While, on the other hand, the students in the scientific visualisation course, who had such tutorial, were generally more positive to using tutorials than not having access to them. Therefore, it seems obvious to make the conclusion that a tutorial will not only help students to be creative and motivate them to investigate further, but a good tutorial will also make the students more positive to using it.

The aim is that the tutorial shall be extended and improved so that it can be an even more valuable tool for the visualisation courses and workshops given within SeSE and UPPMAX. For the computer graphics course it will be investigated whether some of the already existing online tutorials could be used, since constructing such a tutorial is a rather large undertaking.

References

[1] E. Angel, D. Shreiner: *Interactive Computer Graphics: A Top-Down Approach with Shader-Based OpenGL*. 6th ed., Addison-Wesley Publishing Comp., 2011.

[2] M. Ben-Ari: *Constructivism in computer science education*. SIGCSE Bull. **30**(1), 257–261 (1998).

[3] K.E. Boyer, R. Phillips, M.D. Wallis, M.A. Vouk, J.C. Lester: *Investigating the role of student motivation in computer science education through one-on-one tutoring*. Computer Science Education **19**(2), 111–135 (2009).

[4] R. De Villiers: *Usability evaluation of an e-learning tutorial: Criteria, questions and case study*. Proc. 2004 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists on IT Research in Developing Countries (SAICSIT '04), Republic of South Africa, 2004, pp. 284–291.

[5] R. Duit: *The constructivist view: A fashionable and fruitful paradigm for science education research and practice*. In L.P. Steffe, J. Gale (eds.): *Constructivism in Education*, Taylor & Francis, 2012, chapter 14, pp. 271–285.

[6] L. Faessler, H. Hinterberger, M. Dahinden, M. Wyss: *Evaluating student motivation in constructivistic, problem based introductory computer science courses*. In *E-Learn 2006,* World Conference on ELearning in Corporate, Government, Healthcare, & Higher Education, 2006, pp. 1178–1185.

[7] W.A. Hoover: *The practice implications of constructivism*. SEDL Letter **IX**(3), (1996).

[8] M. Knobelsdorf, R. Romeike: *Creativity as a pathway to computer science*. SIGCSE Bull. **40**(3), 286–290 (2008).

[9] M. Knobelsdorf, C. Schulte: *Computer science in context – pathways to computer science*. In R. Lister, S. Lissim (eds.): *Seventh Baltic Sea Conference on Computing Education Research (Koli Calling 2007)*, vol. 88 of CRPIT, Koli National Park, Finland, 2007, pp. 65–76.

[10] D.A. Norman: *What goes on in the mind of the learner*. New Directions for Teaching and Learning **1980**(2), 37–49 (1980).

[11] E. Roberts: *Strategies for encouraging individual achievement in introductory computer science courses*. SIGCSE Bull. **32**(1), 295–299 (2000).

[12] R. Romeike: *Towards students' motivation and interest: Teaching tips for applying creativity*. In Proc. 8th International Conference on Computing Education Research (Koli '08), New York 2008, pp. 113–114.

[13] W.J. Schroeder, K.M. Martin, W.E. Lorensen: *The Visualization Toolkit (4th ed.): An Object-oriented Approach to 3D Graphics*. Pearson Education Inc., 2006.

[14] Swedish escience education, 2013 (Accessed June 14, 2014).

[15] Swedish national infrastructure for computing, 2014 (Accessed June 14, 2014).

[16] Syllabus, Uppsala University, 2010 (Accessed June 14, 2014).

[17] Syllabus, Uppsala University, 2010 (Accessed June 14, 2014).

[18] Studentportalen, Uppsala University, 2014 (Accessed June 14, 2014).

[19] Uppmax, VTK Tutorial, 2014 (Accessed June 14, 2014).

[20] Uppsala Multidisciplinary Center for Advanced Computational Science, 2014 (Accessed June 14, 2014).

[21] T. VanDeGrift: *Encouraging creativity in introductory computer science programming assignments*. In *ASEE, 2007 Annual Conference & Exposition*, 2007. Session: Programming for Engineering Students.

Received August 5, 2014; final form November 24, 2014